

AFIT/GCS/ENG/99M-03

PROTEIN STRUCTURE PREDICTION
USING
PARALLEL
LINKAGE INVESTIGATING GENETIC ALGORITHMS
THESIS

Karl Raphael Deerman, Captain, USAF

AFIT/GCS/ENG/99M-03

Approved for public release; distribution unlimited.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

DTIC QUALITY INSPECTED 2

19990409 045

PROTEIN STRUCTURE PREDICTION
USING
PARALLEL LINKAGE INVESTIGATING GENETIC ALGORITHMS

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirement for the Degree of
Master of Science in Computer Science

Karl R. Deerman, B.S.

Captain, USAF

March 1999

Approved for public release; distribution unlimited.
(A thesis formerly known as FRED.)

PROTEIN STRUCTURE PREDICTION
USING
PARALLEL LINKAGE INVESTIGATING GENETIC ALGORITHMS

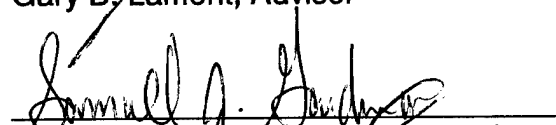
Karl R. Deerman, B.S.
Captain, USAF

Approved:



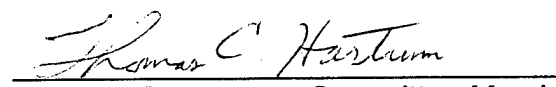
Gary B. Lamont, Advisor

15 MARCH 1999
date



Samuel Gardner, Committee Member

29 MAR 99
date



Thomas C. Hartum, Committee Member

19 Mar 1999
date

Acknowledgments

I could not have accomplished the daunting task of completing this mammoth effort without the continual support of several individuals. My sincerest thanks go out to my thesis advisor, Dr. Gary Lamont, who never balked at my continual pleading for more time. His educational perspectives motivated me to learn more than I thought possible in just 18 months! I'm also indebted to my sponsor, Dr. Ruth Pachter. She never seemed to become frustrated with my inability to fully comprehend molecular biochemistry. I would also like to acknowledge other members of my committee: Dr. Tom Hartrum for teaching me to appreciate the beauty of a well-documented program, Captain Sam Gardner for correcting my math, and Captain Larry Merkle who made me review more chemistry than a computer scientist should have to.

Several students at AFIT have been more than helpful during this investigation. Captain Dave Van Veldhuizen endured my endless questions about genetic algorithms. To his realignment of my genetic algorithm perspectives I will always be indebted. I would also like to thank Captain Lonnie Hammack, Captain Chris Bohn, and Captain Fernando Silva for keeping me non-postal during the long hours of slaving in the lab, and their ability to listen to my endless whining about software that would not perform.

Finally, none of this work would have been possible without the love and support from my wife, Tanya. This thesis is as much your document as it is mine. I know and understand that your sacrifices have been far greater than mine! Thank you, for your patience, devotion, and understanding. I would like to dedicate this document to my unborn child. May one day he be able to read it, comprehend it, and point out the errors his old man has made.

RICERCAR

Karl Raphael Deerman

23 March 1999

Table of Contents

Acknowledgments	i
Table of Contents	ii
List of Equations	v
List of Figures	vi
List of Tables	viii
Abstract	ix
1.0 Introduction	1
1.1 Protein Structure Prediction (PSP) / Protein Folding Prediction (PFP).....	1
1.1.1 PSP Problem Class	2
1.1.2 PSP Structure	3
1.2 Genetic Algorithms	6
1.3 Parallel & Distributed Computing	7
1.4 Visualization	8
1.5 Research Objectives & Rationale	9
1.6 Methodology.....	11
1.7 Assumptions on Research Context.....	11
1.8 Overview	11
2.0 Literature Review	13
2.1 Introduction.....	13
2.2 PSP Landscape.....	13
2.3 PSP Energy Models.....	16
2.4 Bounding the Search Space	20
2.5 Visualizing the Search Space	28
2.6 Summary	31
3.0 Linkage Investigating Genetic Algorithms (LIGAs)	33
3.1 Introduction.....	33
3.2 Problems with the SGA.....	33
3.3 Survey of Current LIGAs	34
3.3.1 Messy Genetic Algorithm (mGA)	35
3.3.1.1 Chromosome Representation	35
3.3.1.2 mGA Algorithmic Phases.....	36
3.4 Selfish Gene Genetic Algorithm (SG GA)	38
3.4.1 SG GA Chromosome Representation.....	40
3.4.2 SG GA Algorithmic Phases.....	40
3.5 Gene Expression Messy Genetic Algorithm (GEMGA)	42
3.5.1 Chromosome Representation	43
3.5.2 Algorithmic Phases.....	44
3.6 Linkage Learning Genetic Algorithm (LLGA)	46
3.6.1 LLGA Chromosome Representation	48
3.6.2 LLGA Algorithmic Phases.....	49
3.7 Comparison	52
3.7.1 Initial Population Explosion Problem.....	52
3.7.2 Algorithmic Complexity	54
3.7.3 Linkage Order Ability	56
3.7 Summary	58
4.0 Genetic Algorithm (GA) Design and Implementation	60
4.1 Introduction.....	60
4.2 CHARMm Implementation Design	61
4.2.1 Integrating CHARMm.....	62

4.2.2 Design Implementation	62
4.3 LLGA/PSP Design and Implementation Details	66
4.3.1 Challenge 1: Building Block Assumption	67
4.3.2 Challenge 2: Recording the Optimal Solution Uncovered	68
4.3.3 Challenge 3: Integration of CHARMM	69
4.3.4 Challenge 4: Parallel Implementation and Execution Timing	69
4.3.4 Challenge 5: Random Number Generator Correctness	70
4.4 AFIT CHARMM Inclusion of Ramachandran Constraints	74
4.5 Summary	76
5.0 Design of Experiments	77
5.1 Test Molecules	78
5.2 General Data Requirements	81
5.2.1 General Data Requirements	81
5.2.2 Random Number Seeds	83
5.2.3 Root Mean Squared Deviation (RMSD)	83
5.2.4 Test Platforms	84
5.3 Experiment Specifics	84
5.3.1 Experiment 1: Parallel vs. Sequential LLGA	85
5.3.2 Experiment 2: Constrained vs. Non-constrained Sequential LLGA	86
5.3.3 Experiment 3: Constrained Parallel LLGA vs. Non-constrained Parallel LLGA	87
5.3.4 Experiment 4: Constrained-pLLGA vs. Constrained Para-REGAL Implementation	88
5.3.5 Experiment 5: Constrained-LLGA, Non-constrained LLGA, pLLGA, constrained pLLGA vs. fmGA	88
5.4 Summary	89
6.0 Results and Analysis	91
6.1 Experiment 1: Parallel vs. Sequential LLGA	91
6.3 Experiment 2: Constrained vs. Non-constrained Sequential LLGA	96
6.4 Experiment 3: Constrained Parallel LLGA vs. Non-constrained Parallel LLGA	99
6.5 Experiment 4: Constrained-pLLGA vs. Constrained Para-REGAL Implementation	102
6.6 Experiment 5: Constrained-LLGA, Non-constrained LLGA, pLLGA, constrained pLLGA vs. pfmGA	103
6.6 LLGA Observations	104
6.7 Energy Landscape Visualization	113
6.8 Summary	117
7.0 Conclusion and Recommendations	118
7.1 Conclusions	118
7.2 Contributions	119
7.3 Recommendations	119
Appendix A. Background on the Protein Folding and Protein Structure Prediction Problems	121
A.1 Introduction to Proteins and Associated Terminology	121
Appendix B. Current Methods for Protein Structure Prediction	125
B.1 Introduction	125
B.2 Practical Methods for Calculating a Protein's Native Structure	125
B.2.1 X-ray Crystallography	126
B.2.2 Nuclear Magnetic Resonance Spectroscopy	128
B.2.3 Computational Models	129
B.2.3.1 Empirical	129

B.2.3.1.1 Anatomy of a Molecular Mechanics Force-Field	130
B.2.3.1.1.1 Bond Stretching Energy.....	131
B.2.3.1.1.2 Angle Bending Energy.....	132
B.2.3.1.1.3 Non-Bonded Energy	132
B.2.3.1.1.4 Torsion Energy	133
B.2.3.2 Semi-Empirical	134
B.2.3.3 Ab Initio	136
B.3 Summary.....	139
Appendix C: Parallelization Techniques.....	140
C.1 Decomposition Techniques.....	140
C.2 Scheduling Strategies	141
C.3 Load Balancing	142
C.4 Introduction to UNITY	143
C.4.1 Explicit Expression of Parallelism	143
C.4.2 Extraction of Proofs.....	144
C.4.3 Mapping of the Initial Design	144
Appendix D: Decomposition of the PSP Problems.....	147
D.1 Decomposition of PSP Problem	147
D.2 Scheduling and Load Balancing of the GAs	149
D.3 Scheduling and Load Balancing of CHARMM	153
D.4 Summary	155
Appendix E: Additional LIGA Structure	156
E.1 Messy Genetic Algorithm (mGA)	156
E.1a mGA Chromosome Representation: Positional Precedence	156
E.1b mGA Chromosome Representation: the Competitive Template	156
E.1c mGA Algorithmic Phases: partially enumerative initialization (PEI).....	157
E.1d mGA Algorithmic Phases: Cut-and-Splice	159
E.2 Selfish Gene Genetic Algorithm (SG GA).....	161
E.2a SG GA Chromosome Representation: Virtual Gene Pool Growth Rate.....	162
E.2b SG GA Algorithmic Phases: Mutation.....	163
E.2c SG GA Algorithmic Phases: Allele Frequencies and Epsilon (ϵ)	164
E.3 Gene Expression Messy Genetic Algorithm (GEMGA).....	164
E.3a Chromosome Representation: Gene Representation.....	165
E.3b Algorithmic Phases: Initialization	165
E.3c Algorithmic Phases: RecombinationExpression.....	166
E.4 Linkage Learning Genetic Algorithm (LLGA)	168
E.4a LLGA Algorithmic Phases: Exchange Operator.....	168
E.4b LLGA Algorithmic Phases: Preconvergence Avoidance and Introns.....	170
Appendix F: Software Locations.....	172
F.1 Source Code	172
F.2 Input and Output Files	172
Appendix G: Flow Diagrams for AFIT's Implementation of the CHARMM Energy Model.....	175
Appendix H: Object Classes for the Redesigned LLGA	182
Appendix I: Newman Projection for the Each Dihedral Constraint	185
Appendix J: Statistics Explained	186
J.1 Analysis of Variance Testing (ANOVA).....	186
J.2 Kruskal-Wallis H Test	188
J.3 The Central Limit Theorem	189
Appendix K. Material for Test Platforms.....	190

K.1 AFIT Network Of Workstations (NOW)	190
K.2 IBM SP2.....	190
K.3 AFIT Heterogeneous Beowulf	191
8.0 Vita	192
9.0 References	193

List of Equations

Equation 1: Simplified Assumption	4
Equation 2: Psi , Phi, and Chi Constraints	6
Equation 3: Mathamatical Representation of NC Problem.....	8
Equation 4: Size of the Landscape	13
Equation 5: Bond & Angle Energy Equations	17
Equation 6: Torsion Potential	17
Equation 7: Improper Torsion	17
Equation 8: Lennard-Jones Potential.....	18
Equation 9: Hydrogen Bonding Energy Reduction.....	18
Equation 10: Water-Water Interaction	18
Equation 11: Complete CHARMM Energy Equation.....	18
Equation 12: AFIT's CHARMM Implementation	19
Equation 13: 1 st Coordinate Transformation	26
Equation 14: 2 nd Coordinate Transformation	27
Equation 15: p-norm.....	29
Equation 16: Selfish Gene Model of Mutation.....	41
Equation 17: SG Steady State.....	41
Equation 18: GEMGA Population Requirement.....	44
Equation 19 : LLGA Building Block Linkage	47
Equation 20: RMSD Calculation	83
Equation 21: Total Overhead.....	92
Equation 22: Speedup.....	93
Equation 23: Efficiency.....	95
Equation 24: Simplified Semi-Empirical Energy Equation.....	130
Equation 25: Bond Stretching Energy.....	131
Equation 26: Angle Bending Energy.....	132
Equation 27: Non-bonded Energy Equation	133
Equation 28: Torsion Energy Equation	134
Equation 29: Schrodinger's Equation	135
Equation 30: Reduction of Schrodinger's Equation.....	135
Equation 31: Simplified Schrodinger's Equation	135
Equation 32: Schrodinger's Equation for a Molecule	136
Equation 33: Fock Operator	136
Equation 34: Hamiltonian, Coulomb, and Exchange Operators.....	137
Equation 35: Minimum Energy Equation.....	137
Equation 36: Definition of Matrix Elements	137
Equation 37: Hartree-Fock Equation	137
Equation 38: Spatial Orbitals	138
Equation 39: Roothaan Equations	138
Equation 40: Overlap and Fock Matrices.....	138
Equation 41 : PEI Population Equation.....	158
Equation 42: Cut Probability	160

Equation 43: Introns Required per X Exons	170
Equation 44: Two-Way ANOVA Design.....	187
Equation 45: Kruskal-Wallis H Test.....	188

List of Figures

Figure 1: A Three Amino Acid Protein	4
Figure 2: PFP Simplified.....	5
Figure 3: Metastable State	14
Figure 4: The <i>Glassy</i> Funnel	15
Figure 5: Partial Double-Bond Characteristics.....	21
Figure 6: The <i>trans</i> Formation.....	22
Figure 7: The <i>cis</i> Formation	22
Figure 8: Original Ramachandran Plot	24
Figure 9: Stryer's Ramachandran Plot.....	25
Figure 10: Applying 1st Transform	26
Figure 11: Repositioning the Origin	26
Figure 12: Φ -Axis Transformation	27
Figure 13: P-Norm Comparison.....	30
Figure 14: Energy Landscape Visualization.....	31
Figure 15: mGA Positional Precedence at Work	36
Figure 16: Propagation of a Chromosome Fragment	39
Figure 17: Steps of SEARCH	43
Figure 18: GEMGA Chromosome Representation.....	44
Figure 19: Harik's Linkage Definition	48
Figure 20: Visualization of a Chromosome	49
Figure 21: Deletion Process Tightening of Building Blocks.....	50
Figure 22: Crossover Operation Tightening of Building Blocks.....	51
Figure 23: LIGA Complexity Curves	56
Figure 24: AGCT Genetic Algorithm Toolkit	61
Figure 25: CHARMm Source Code Control Flow.....	63
Figure 26: Z-matrix Format.....	63
Figure 27: LLGA Class Hierarchy.....	67
Figure 28: Uniformly Distributed Random Numbers	72
Figure 29: Autocorrelation for Random Seed 1	73
Figure 30: Orginal Chromosome Decoding	74
Figure 31: Modified Chromosome Decoding	75
Figure 32: Extended Conformation of [Met]-Enkephalin	79
Figure 33: Extended Conformation of Polyalanine.....	80
Figure 34: pLLGA Speedup.....	94
Figure 35: pLLGA Efficiency.....	96
Figure 36: Energy Characteristics of the cLLGA.....	98
Figure 37: Energy Characteristics of the LLGA	99
Figure 38: Speedup Comparison between pLLGA and cpLLGA.....	101
Figure 39: Efficiency Comparison between the pLLGA and cpLLGA.....	102
Figure 40: Comparison Between Linkage Investigating Gas	104
Figure 41: BBs Uncovered and Maintained for the pLLGA and Random Seed 1	105
Figure 42: BBs Uncovered and Maintained for the pLLGA and Random Seed 2.....	106
Figure 43: BBs Uncovered and Maintained for the pLLGA and Random Seed 3.....	107
Figure 44: BBs Uncovered and Maintained for the cpLLGA and Random Seed 1	108

Figure 45: BBs Uncovered and Maintained for the cpLLGA and Random Seed 2	109
Figure 46: BBs Uncovered and Maintained for the cpLLGA and Random Seed 3	110
Figure 47: BBs Uncovered and Maintained for the cpLLGA, Random Seed 3, and the New Selection Operator.....	111
Figure 48: Energy Characteristics of the New Selection Operator.....	112
Figure 49: Energy Characteristics of the New Selection Operator (smaller scale)	113
Figure 50: Energy Landscape Visualization.....	115
Figure 51: Condensed Energy Landscape Visualization.....	116
Figure 52: X-ray Diffraction Pattern for protein Lac Repressor [27]	127
Figure 53: Electron Density Map [29]	127
Figure 54: Sample 1D NMR Spectrum [26]	129
Figure 55: Overview of the Mechanical Molecular Model Forces	131
Figure 56: Bond Stretching.....	131
Figure 57: Angle Bending.....	132
Figure 58: Non-bonded Interaction.....	133
Figure 59: Torsion Energy	134
Figure 60: Direct LLGA Task Decomposition Schedule.....	149
Figure 61: Direct fmGA Task Decomposition Schedule.....	149
Figure 62: Zhu's Scheduling for LLGA.....	150
Figure 63: Zhu's Scheduling for fmGA	151
Figure 64: Kruatrachue-Lewis' Duplicate Scheduling for LLGA	152
Figure 65: Kruatrachue-Lewis' Duplicate Scheduling for mGA	152
Figure 66: Two Simple Parallelizations of CHARMM	153
Figure 67: Zhu's CHARMM Schedule.....	154
Figure 68: Kruatrachue-Lewis' CHARMM Schedule	155
Figure 69: mGA's Inter-Chromosomal Dominance Operator	156
Figure 70: Population Growth Rate for the mGA	159
Figure 71: Cut-and-Splice in a Nutshell	161
Figure 72: Population Growth Rate for the SG GA	163
Figure 73: 2D Fitness Landscape.....	164
Figure 74: GEMGA Population Growth vs. Increasing Initial Variance.....	166
Figure 75: An Overview of the Exchange Operator	169
Figure 76: Intron Requirement Trend	171
Figure 77: Sample LLGA Input File	173
Figure 78: Top Level Structure Chart	175
Figure 79: CHARMM Top-level Data and Structure Diagram	176
Figure 80: Non-bonded Energy Module Expansion	177
Figure 81: New Coordinates Module Expansion.....	178
Figure 82: Local Minimization Top-Level Data and Structure Diagram	179
Figure 83: Module dlinmin Expansion.....	180
Figure 84: Module mymnbrak Expansion	180
Figure 85: Module dbrent Expansion	181
Figure 86: Phi Constraints.....	185
Figure 87: Glycine Phi Constraints	185
Figure 88: Psi Constraints	185
Figure 89: Omega Constraints	185
Figure 90: Chi Constraints.....	185

List of Tables

Table 1: Practical Differences of the Computational Engines	3
Table 2: Comparison Between Commonly used Energy Functions	20
Table 3: Bond Angle Conventions	23
Table 4: Comparison of van der Waals Contact Distances.....	23
Table 5: Loose Constraints for [Met]-Enkephalin	28
Table 6: Loose Constraints for Polyalanine	28
Table 7: Visualization Legend	30
Table 8: Algorithmic Complexity	55
Table 9: Algorithm Characterisitics Summerized	58
Table 10: Successful LIGA Applications.....	59
Table 11: Correct Energy Values Associated with the Correct Z-Matrix File	64
Table 12: Chromosome Decoding	65
Table 13: Pairwise Correlation.....	73
Table 14: Dihedral Angles for [Met]-Enkephalin Accepted Energy Minimum	78
Table 15: Dihedral Angles for Polyalanine Accepted Energy Minimum.....	80
Table 16: General Data Requirements for Each Experiment.....	82
Table 17: Random Number Seeds Use In Initialization.....	83
Table 18: Processor Allocation per Architecture	84
Table 19: General LLGA Parameters	85
Table 20: Parameters for Experiment 1	86
Table 21: Parameters for Experiment 2.....	87
Table 22: Parameters for Experiment 4.....	88
Table 23: Parameters for Experiment 5.....	89
Table 24: End of Execution Energy Comparison bewteen LLGA and pLLGA	91
Table 25: pLLGA Average Execution Times - ABC Beowulf	92
Table 26: pLLGA Average Execution Times - Maui SP2	92
Table 27: Statisticals For pLLGA Implementations	95
Table 28: Constrained vs. Non-Constrained.....	96
Table 29: Final Energy Characteristics for the LLGa and cLLGA	97
Table 30: Total Overhead for cpLLGA and pLLGA.....	100
Table 31: Limits of the Landscape Visualization	114
Table 32: Dihedral Angles for Molecule One of the Visualization.....	114
Table 33: Dihedral Angles for the Last Molecule of the Visualization.....	114
Table 34: Possible Fitness Functions	120
Table 35: Phi & Psi Pairs of Common Secondary Structures	122
Table 36: Enumeration Time of 1.3×10^{30} Search Space at One Solution per Clock Cycle	124
Table 37: Principles of UNITY	144
Table 38: Steps to a UNITY Proof	144
Table 39: Mapping to Asynchronous Shared-Memory Architectures	145
Table 40: Mapping to Distributed Systems	146
Table 41: Mapping to Synchronous Architectures	146
Table 42: LLGA Task Decomposition	147
Table 43: mGA Task Decomposition	148
Table 44: CHARMM Decomposition	148
Table 45 : Cut-and-Splice Combination Possibilities.....	160
Table 46: Example ANOVA Table	187
Table 47: Two-Way ANOVA Decomposition Table	187

Abstract

AFIT has had a long-standing interest in solving the protein structure prediction (PSP) problem. The PSP problem is an intractable problem that if "solved" can lead to revolutionary new techniques for everything from the development of new medicines to optical computer switches. The challenge is to find a reliable and consistent method of predicting the 3-dimensional structure of a protein given its defining sequence of amino acids. PSP is primarily concerned with predicting the tertiary protein structure without regards to how the protein came to this folded state. The tertiary structure determines the protein's functionality.

Genetic algorithms (GAs) are stochastic search routines that are capable of providing solutions to intractable problems. The use of GAs plays an important part in the search for near optimal solutions in large search spaces. The PSP solution landscape is so large and complex that deterministic methods flounder due to the combinatoric issues involved with enumerating these massive search spaces. This makes the GA an ideal candidate for finding solutions to the PSP problem.

This is an engineering investigation into the effectiveness and efficiency of the Linkage Learning GA (LLGA) applied to the PSP problem. The LLGA implementations takes explicit advantage of "tight linkages" early enough in its algorithmic processing to overcome the disruptive effects of crossover. The LLGA is integrated with the previously developed and tested AFIT CHARMM energy model software.

Furthermore, a parallel version, pLLGA, is developed using a data partitioning scheme to "farm out" the CHARMM evaluations. Portability across AFIT's heterogeneous ABC Beowulf system, distributed networks, and massively parallel platforms is accomplished through the use of object-oriented C++ and the Message Passing Interface (MPI). This model improves the efficiency of the LLGA algorithm.

Ramachandran developed constraints are incorporated into the LLGA to exploit domain knowledge in order to improve the effectiveness of the search technique. This approach, constrained-LLGA (cLLGA), has been parallelized using the same decomposition as the pLLGA. This new implementation is called the constrained-parallel LLGA (cpLLGA). Efficiency analysis for these two implementations is discussed.

Finally, the results from these experiments are compared to previous AFIT implementations. The parallel fast messy GA and the parallel real-valued GA are compared to the pLLGA and cpLLGA, respectively.

1.0 Introduction

Computer solutions to many complex optimization problems cannot be obtained in acceptable amounts of time. Even the most powerful of today's super computers, if given a problem of sufficient complexity, would take centuries to return a solution¹. Yet, if truly complex problems – such as the so-called “Grand Challenges” – are to be solved, improved algorithmic methods must be accompanied by similar improvements in computer technology.

Recent research efforts have turned towards creating general search algorithms designed to “overpower” these difficult problems by finding “acceptable” solutions. The Air Force Institute of Technology (AFIT) has long been a leader in the pursuit of solving these Grand Challenges. Two main research efforts spearheaded by AFIT as part of these efforts are parallel computing and semi-optimal search algorithms. Parallel computing is a field of computer science/engineering that transforms problems traditionally solved sequentially by decomposing them into independent subproblems that can be solved simultaneously on separate processors. The other effort, semi-optimal stochastic search algorithms are a means to find reasonably “good” or suboptimal solutions to intractable problems.

One such problem AFIT has had a long interest in solving is the protein structure prediction (PSP) problem. The PSP problem is an intractable problem contained in the class of Grand Challenges [47]. The challenge is to find a reliable and consistent method of predicting the 3-dimensional structure of a protein given its defining sequence of amino acids. An exhaustive search of a reasonable discretization of the entire solution space for even the smallest proteins consumes more time than the estimated age of the universe [15]! This thesis research explores genetic algorithms as a possible means to solve the PSP problem.

1.1 Protein Structure Prediction (PSP) / Protein Folding Prediction (PFP)

The PSP and Protein Folding Prediction (PFP) problem are two related problems that have the same overall objective: to accurately predict the conformational structure of a protein. The PSP approach is primarily concerned with predicting the protein's tertiary structure without regards to how the protein arrived at this folded state. On the

¹ See Appendix A.

other hand, the PFP's primary concern is the transition process the protein undergoes starting from the primary structure and ending in the tertiary structure (*ab initio*).

A protein is comprised of amino acids linked together through chemical bonding. The tertiary structure of a protein corresponds to positioning all its amino groups in such a manner that the overall molecule has the lowest energy (i.e., conformational energy). Although the structure of a specific protein, hence the positions of all the atoms, can be accurately determined using currently available methods (x-ray crystallography and nuclear magnetic resonance – see **APPENDIX B** for a detailed description –), each of these methods requires several years to obtain results for a single protein [18], and the protein must first be synthesized or isolated. Therefore, a portion of the PSP/PFP community has turned its attention to predicting the conformational structure through the use of computer models and simulations.

1.1.1 PSP Problem Class

The PSP problem belongs to the class of problems for which currently there is no known non-polynomial-time complexity nondeterministic algorithm (i.e., NP-complete) [48]. Proving the problem is in NP and it can be mapped to some other “known” NP-complete problem shows its NP-completeness. The PSP problem is in NP because the size of the search space is defined by the number of independently variable dihedral angles raised to the power of the number of allowed rotation angles. In other words, (cardinality |Protein|)ⁿ where n = number of positions a dihedral can hold and the cardinality represents the number of independently variable dihedral angles within the protein. For example, if we used a three-peptide protein with p₁ rooted at the axis of a normal Cartesian plane (0,0), p₂ can be positioned at 0° to 360° about the origin, and p₃ can similarly rotate about p₂. In this example, P = {p₁, p₂, p₃} and n = 360 yielding 2³⁶⁰ possibilities. This simplification allows for more variability within the rotation than what is allowed by nature, which is explained in more detail in the following section.

Mapping the PSP problem to another known NP-complete class problem is a trivial but rather lengthy matter. The requirement is to prove that the PSP problem is polynomial transformable to another known NP-complete problem. The argument and proof is in [48]. Because the PSP problem is NP-complete, we will never truly be able to say, “Eureka, we found the global optimal answer!” through computational power alone.

1.1.2 PSP Structure

The structure of the PSP problem is exclusively defined by the means of calculating the conformational energy. For instance, usually empirical methods only take dihedral bond angles into account while holding such energy terms like bond length and bond angle constant. Furthermore, they usually don't account for interactions between the protein and the surrounding solvent. On the other end of the calculation spectrum, *ab initio* methods use all atomic interactions when calculating the conformational energy as the protein folds. **Table 1** lists the practical differences between the three general computational methods.

Empirical Methods

- Used in molecules containing thousands of atoms.
- Can be applied to organics, oligonucleotides, peptides, and saccharides (metallo-organics & inorganics in some cases).
- Vacuum, implicit, or explicit solvent environments
- Can only be used to study ground state.
- Can be used to explain thermodynamic and kinetic properties.

Semi-Empirical Methods

- Limited to hundreds of atoms
- Can be applied to organics, organo-metallics, and small oligomers (peptide, nucleotide, saccharides).
- Can be used to study ground, transition, and exited states (certain methods).

Ab Initio Methods

- Limited to tens of atoms and still best performed using a supercomputer.
- Can be applied to organics, organo-metallics, and molecular fragments (e.g. catalytic components of enzyme).
- Can be used to study ground, transition, and exited states (certain methods).

Table 1: Practical Differences of the Computational Engines

Assuming the use of an empirical computational method, the PSP problem structure reduces to:

Given a protein (P) comprised of a chain of peptides (p), each having a dihedral angle associated on one end, given the position of the first peptide find the positions of $\{p_1, p_2, \dots, p_n\}$ such that the conformational energy level (C) is the lowest."

$$\exists P : \min \left| \sum_{i=1}^n C(p_i) \right|$$

Equation 1: Simplified Assumption

Of course, this is a drastic over simplification of the PSP problem because there are many molecular chemistry concepts that we have not yet taken into account. The earlier assumption that a peptide only has a single dihedral angle is intuitively wrong. We know from basic geometry that to position an object in 3D space we must use three angles to define rotation about the x, y, and z-axis. Thus, in molecular chemistry, a polypeptide is viewed from the description of a single peptide unit. The polypeptide has a direction associated with it for geometric and scientific description purposes. The peptide begins at the α -amino and ends at the α -carboxyl group. (Refer to **Figure 1**.) Each peptide has three associated angles: Ψ (*psi*), ϕ (*phi*), and ω (*omega*).

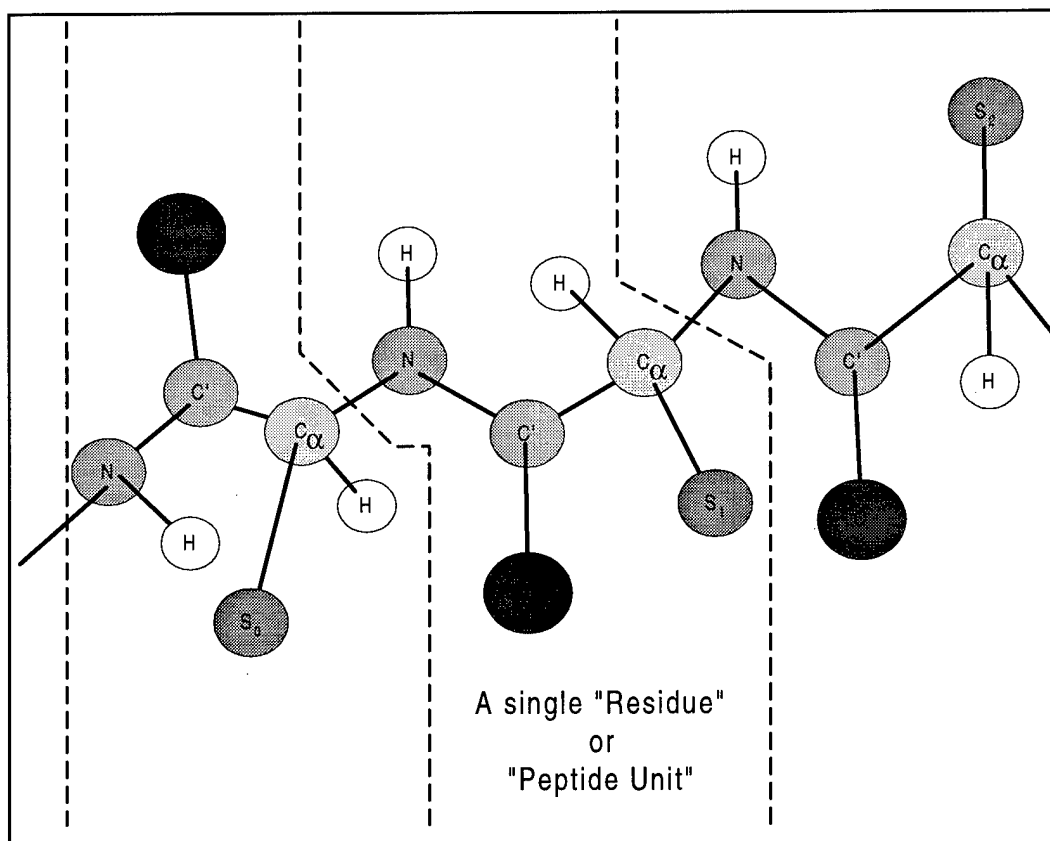


Figure 1: A Three Amino Acid Protein

A peptide can be considered rigid and planar about the ω dihedral angle², but this simplifying assumption still allows each peptide structure to rotate at either side of the α -carbon because these bonds are pure “single” bonds. Rotations about these bonds are defined as Ψ (*psi*) and ϕ (*phi*) dihedral angles. Looking at **Figure 2**, Ψ refers to the angle of rotation of the plane on the left about the C-C $_{\alpha}$ single bond and ϕ refers to the angle of rotation of the plane on the right about the C $_{\alpha}$ -N single bond. [22].

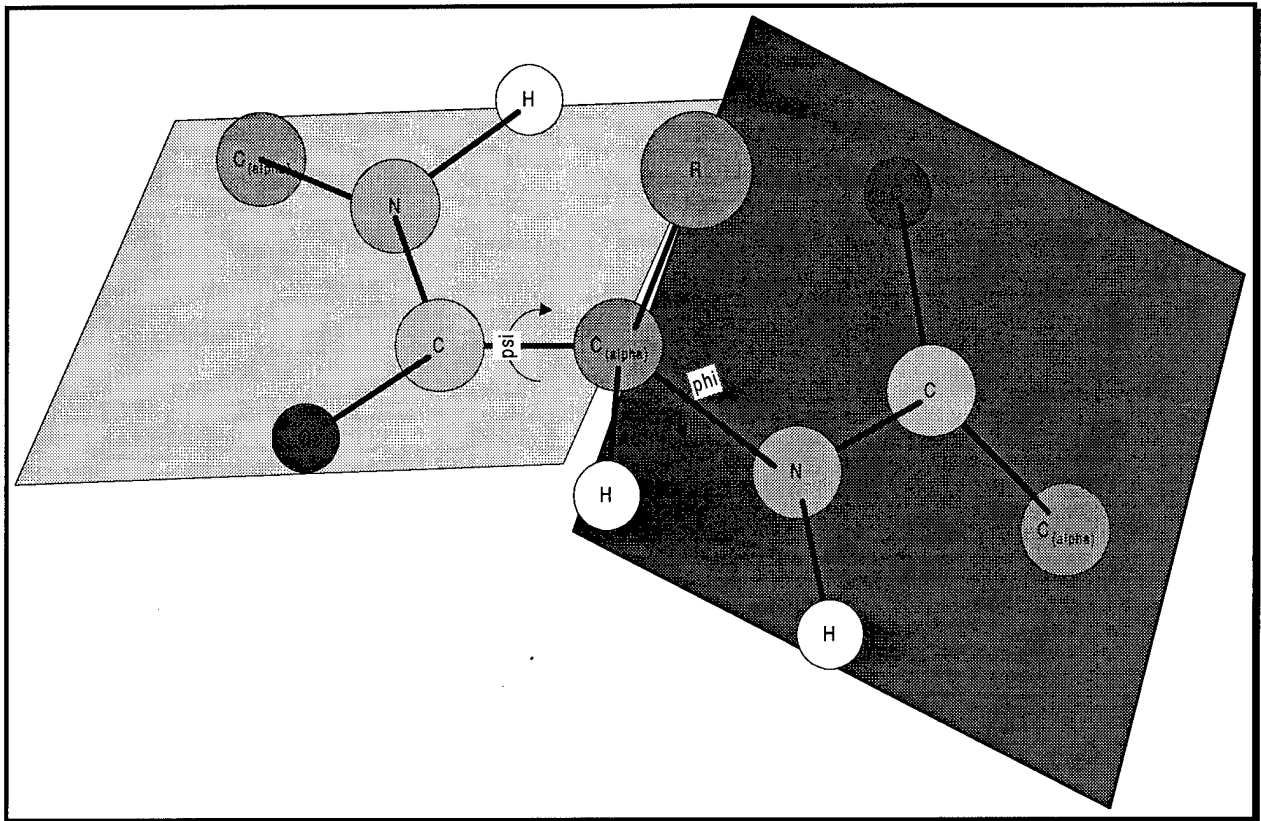


Figure 2: PFP Simplified

This methodology simplifies the protein molecule's conformation energy calculation to just the dihedral angle specified for each peptide. It does not take into account the rotations of protein side-chains.

So far, we have discussed how each of the peptides connects in order to form the overall protein. This is called the molecular backbone of the protein, and it is distinguished by regular repeating amino acid sequences, but the protein also contains side-chains. Side-chains are distinctive non-repeating chemical structures [22]. **Figure 1** and **Figure 2** indicate the side-chains as a single R atom, but they can be comprised

² ω is held to 180°

of many atoms and vary in length. The dihedral angle(s) formed by the side-chains are designated by χ (*chi*). Each protein can have many χ dihedral angles. **Equation 2** indicates the basic data requirements per peptide to completely define the PSP problem's structure.

$$\begin{aligned}\Psi &= \langle 0 - 360 \rangle \\ \Phi &= \langle 0 - 360 \rangle \\ \chi_i &= \langle 0 - 360 \rangle\end{aligned}$$

Equation 2: Psi , Phi, and Chi Constraints

Of course, there are still other requirements to meet, but these “requirements” only serve to constrain the possible positions these angles can hold (i.e., no two atoms can occupy the same space, two bonded atoms cannot be separated by more than a few angstroms, etc.). Refer to [14, 15, 18, and 37] for a more complete discussion. It is sufficient for our discussion here to understand that the positions of each atom within the protein molecule can be defined by **Equation 2**.

1.2 Genetic Algorithms

Genetic algorithms (GAs) are semi-optimal stochastic search algorithms that are capable of providing “good” solutions to intractable problems. In practical applications, the execution time of a genetic algorithm is typically dominated by the fitness function calculation. This function is problem domain dependent and usually of polynomial time complexity. The use of GAs plays an important part in the search for near optimal solutions in large search spaces. Such search spaces (landscapes) are so large or complex that deterministic methods flounder due to the combinatoric issues. The algorithmic power of GAs come from their robustness, and their ability to generally find “acceptably good” solutions to these complex problems “acceptably quickly” [13]. GAs are loosely based on theoretical evolutionary processes [16]. Therefore, many of the terms associated with evolution and biology are interchangeable with terms created specifically for GAs³.

GAs work on populations of solutions called chromosomes. Historically, the first well-known genetic algorithm was the simple GA (sGA) developed by Goldberg. sGAs perform three basic operations on the chromosome populations: selection, crossover, and mutation [15]. The algorithm steps through these three operations repeatedly until

³ Reference Appendix A: Background on Genetic Algorithms by Charles Kaiser

some stopping criterion is met. The execution of a single pass through these steps is called a generation.

The sGA accomplished several important tasks for the genetic algorithm community – the most important was validating the effectiveness and efficiency of GAs in general. Over time, research suggested that a class of problems, called “deceptive problems,” cannot be effectively evaluated by the sGA [1]. This class of problems is characterized by having coding function combinations that have misleading low-order building blocks causing the GA to converge to sub-optimal points. Therefore, some GA researchers ushered in the era of the linkage investigating genetic algorithms (LIGAs) [1, 2, 4, 5, 7, 8, 9]. The algorithmic concentration of this thesis research is on the LIGA family of GAs.

1.3 Parallel & Distributed Computing

GAs are easily parallelizable because one can execute multiple copies of the same GA program with different subpopulations on different processors and select the best solution after the last processor has terminated. Two general forms of parallelism exist which can lead to performance improvements when algorithms are implemented in parallel: data parallelism and control parallelism. These parallelization techniques are discussed in **APPENDIX C**. The properties that can be most profitably exploited depend upon the problem domain, the specific GA algorithm, and the parallel architecture chosen.

A problem is well-parallelized if it can be computed very quickly by an algorithm which uses a feasible amount of processors [44]. A natural way to uncover whether or not a particular algorithm is parallelizable is to determine if it belongs in *Nick’s Class* (NC) [44]. NC contains the class of computational problems that can be solved on the parallel random access machine (PRAM) model by a deterministic algorithm in polylogarithmic time using only a polynomial number of processors. (The PRAM theoretical model of computation is formally defined as a computer consisting of p processors and a global memory of unbounded size that is uniformly accessible to all processors [36].) In general, NC includes algorithms that satisfy:

$$T_s(n) = n^l \log^k n \text{ with}$$

$$P(n) = nl \Rightarrow T_p(n) = \frac{n^l \log^k n}{n^l} = \log^k n$$

where, T_s = sequential execution time,
 T_p = parallel execution time.

Equation 3: Mathamatical Representation of NC Problem

Genetic algorithms are one such class of algorithms that satisfy this NC structure.

A wide variety of parallel architectures have been designed and implemented. High level design options include: *single instruction, multiple data stream* (SIMD) – a single control unit dispatches instructions to each processing unit; *multiple instruction stream, single data stream* (MISD) – each processor performs different operations on a single data stream (i.e., vector processors); *multiple instruction stream, multiple data stream* (MIMD) – each processor is capable of executing a different program independent of the other processors [36]. No single architecture, of course, has been shown to be clearly superior for all applications.

1.4 Visualization

The basic premise of scientific visualization is the use of computer-generated pictures to gain insight from the data [60]. This is still a very active and vital arena of research. In particular, the GA community does not have a solid foundation of visualization techniques. On the other hand, commercial packages for visualizing proteins, polypeptides, and other molecule are readily available (i.e., Quanta, RASMOL, Cerius, etc.). Alas, these commercial packages only allow the user to visualize postpartum. Currently, there is no software package available which allows the user to manipulate the molecule as it folds. This is due to the fact that to view the folding protein the update rate of the visualization would need to be on the order of a femtosecond (e.g., 10^{-15} seconds). On the other hand, our contribution to this vestal area of expression is the ability to visualize the search space traversed by the GA.

As we discussed previously, the PSP search space is massively “huge” when visualized in 2 dimensions. But when viewed in its true n -dimensional form where n equals the number of independent variables entered into the energy function, the search space is drastically reduced being bound in all dimensions by 360° . Except now we have the problem of rendering a 25 dimensional picture for just even for a small protein such as [Met]-Enkephalin! Some research indicates that the resulting image is a 25

dimensional funnel⁴. We have attempted to transform this insurmountable situation into a comprehensible rendering in just 3-dimensions. Our approach takes full advantage of mathematical norms and color to produce an indication of the PSP problem domain landscape and the path in which the GA traversed to find the minimum.

1.5 Research Objectives & Rationale

The goal of this research is to investigate the protein structure prediction problem using the spectrum of Evolutionary Algorithms such as genetic algorithms, evolution strategies, and evolutionary programming. The summation of the research directly contributes to the continuing efforts of the United States Air Force Research Laboratory's (AFRL) search for a robust and efficient technique to expedite their efforts in developing new materials. In particular, AFRL is interested in developing chromophore-substituted polymer chains with control optical properties, so-called smart filters or optical switches [37].

The specific intentions of the research are decomposed into the following objectives:

OBJECTIVE 1: *Investigate the protein structure prediction problem.*

RATIONALE: To understand the problem domain.

- 1) Learn key PSP concepts and terminology.
- 2) Understand scientific limitations that restrict "our" ability to directly measure a protein's folding process.
- 3) Understand different methods for measuring a protein's conformational energy (i.e., x-ray crystallography, nuclear magnetic resonance, computational mechanics).

OBJECTIVE 2: *Investigate the spectrum of Evolutionary Algorithms such as genetic algorithms, evolution strategies, and evolutionary programming.*

RATIONALE: To understand the chosen algorithm domain.

- 1) Develop at least one building block propagating GA based upon analysis (i.e., Selfish Gene GA, Linkage Learning GA, Compressed Linkage Learning GA, or Gene Expression Messy GA).
- 2) Integrate building block propagating GA with the CHARMM energy function for the PSP problem.
- 3) Compare building block propagating GA with fast messy GA (fmGA) currently in AFIT Toolbox, for effectiveness in finding conformational energy states. (sequential model: Are we getting a corrected answer? - effectiveness)

⁴ See Chapter 2.

- 4) Compare building block propagating GA with fmGA for efficiency in finding conformational energy states. (sequential model: Are we getting the answer in comparable time? - efficiency)
- 5) Parallelize building block propagating GA using farming model for fitness evaluations and compare to parallel fmGA (pfmGA) based on effectiveness and efficiency in finding conformational energy states. (parallel model: Are we getting the correct answer in a comparable amount of time?)

OBJECTIVE 3: *Develop a bounding filter using accepted molecular biochemistry concepts in order to curtail the number of solutions possible in an attempt to limit the fitness landscape.*

RATIONALE: AFIT's previous work in this area curtailed our ability to use the product of the research in other Evolutionary Algorithms or different search methodologies.

- 1) Create bounding function using Ramachandran Plots and coordinate transformations using portability as key design consideration [37].

OBJECTIVE 4: *Apply Evolutionary Algorithms integrated with problem domain fitness functions to a variety of test cases (larger proteins -> more atoms) in serial and parallel implementations.*

RATIONALE: AFIT's previous research has been confined to a relatively noncomplex 5-residue protein. The higher complexity associated with "larger" proteins (i.e., 100-residues) must be determined and overcome.

- 1) Port AFIT research efforts to larger proteins.
- 2) Develop a set of procedures for GA integration with larger protein representation.
- 3) Investigate automation.
- 4) Create a USER'S GUIDE for developed building block propagating GA, the current implemented fmGA, and the current implemented pfmGA.

OBJECTIVE 5: *Create effective algorithm visualization methodology to facilitate future PSP and GA researchers.*

RATIONALE: The pattern in which an Evolutionary Algorithm searches a problem domain landscape is inherently difficult to visualize. The product of this research should allow future PSP researchers to "see" the problem domain landscape being uncovered.

- 1) Build GA solution space visualization tools (2D phenotype vs. genotype) - automated
- 2) Build GA search space visualization tools (3D mapping of search area) - automated

- 3) Create and document methodology of electronically porting final protein “answers” produced by GA into VMD or other visualization tools for structural comparison with “actual” conformation.

1.6 Methodology

The existing CHARMM energy model developed by Brinkman [18] and modified by Gates [15] is integrated with the Linkage Learning Genetic Algorithm (LLGA) developed by Harik [17]. It is engineered to incorporate parallel constructs (the parallel LLGA) and Ramachandran constraints (constrained pLLGA & constrained LLGA).

The serial and parallel implementations are compared for correctness and performance gains. Then, these algorithms are compared against previous AFIT results for the fast messy GA [15] and hybrid GA [37]. The constrained implementations are compared against Kaiser’s work involving real value constrained GAs [37]. Finally, the protein conformations uncovered in the GA searches are transformed by our visualization software to produce an image of the traversed search space.

1.7 Assumptions on Research Context

There are several assumptions limiting the research scope as presented in this thesis:

- ◆ The GAs in AFIT’s Genetic Algorithm Toolkit (AGCT) work correctly (see **Figure 24**).
- ◆ The CHARMM energy model implemented by previous master students works correctly and the correct z-matrix, RTF, and parameter files are available.
- ◆ Any software developed is considered “engineering software” and may include design alternatives that do not completely follow sound software engineering principles.
- ◆ The reader has a basic understanding of GAs, computer science, parallelization techniques, and scientific experimentation as an aid to understanding **Chapters 2, 3, 4, and 5**.

These assumptions are included to curtail the scope of this thesis presentation.

1.8 Overview

This chapter has introduced the general research problem (i.e., the PSP/PFP), described the main elements of the approach, and rationalized the need for expanding the research effort on GAs and the PSP problem. The rest of this thesis is decomposed into six areas. **CHAPTER 2** discusses the problem domain associated with the PFP problem, justifies the selected energy model, covers the problem domain/algorithm domain integration, and presents a visualization tool. **CHAPTER 3** explores different linkage investigating genetic algorithms (LIGAs), which we apply towards solving the

PSP problem. **CHAPTER 4** presents the details for parallelization of the selected solution approach. **CHAPTER 5** explains the experimental design, and **CHAPTER 6** analyzes the results of these experiments. Finally, **CHAPTER 7** concludes this thesis and presents direction for future PSP/PFP voyagers.

2.0 Literature Review

2.1 Introduction

Many articles have been written covering the basics of the Protein Structure Prediction (PSP) problem, it is our intention here not to duplicate these efforts [3, 14, 15, 18, 19, 21, 37, 48, 50, 61, 63, 64]. The reader is directed to **APPENDIX A** for the basic PSP background information. The intent of this chapter is to cover areas of the PSP problem that directly impact our efforts. Mainly, the PSP landscape is discussed in **Section 2.2**; **Section 2.3** justifies our energy model selection; **Section 2.4** summarizes our approach to bounding the search space; and our unique visualization technique is reviewed in **Section 2.5**.

2.2 PSP Landscape

The most challenging aspect of the PSP problem is that the conformation energy calculation creates an enormous number of local minima. Therefore, any attempt using local minimization techniques usually becomes caught in arbitrary local minima. These minima can be arbitrarily far from the global minimum. The small differences between the assumed conformation energy and these minima makes it extremely difficult to know how close one is to the accepted energy minimum simply by comparing to the calculated energy. It is assumed that the protein's geometry defined by the naturally occurring conformation is the global minimum [50]. Note that the energy model used and the refinement of the input data required in the model define the size of the energy landscape (i.e., the search space)⁵.

For example, let's assume an energy model that only requires as input the dihedral angles of a protein that consists of 24 dihedral angles. If each dihedral angle were allowed to rotate freely about each bond without considering any constraints, then **Equation 4** would model the size of the conformation space.

$$\text{search space} = d^N$$

where d is the number of values each dihedral angle can assume
 N is the number of independently variable dihedral angles

Equation 4: Size of the Landscape

⁵ See Section 2.3 for a discussion of the different energy models used in PSP calculations.

For argument's sake, we allow two atoms to occupy the same space and let the energy model indicate the invalidity of the resulting protein⁶. Therefore, supposing that each dihedral angle has 360 possible values, our example protein would have approximately 2.25×10^{61} different orientations⁷. If only one tenth of these orientations belonged to the set of possible minima, then there would be 1 out of 2.25×10^{60} chances of randomly finding the global minimum.

The search space terrain of such a protein is extremely rugged consisting of millions of valleys and peaks. How a protein, with no known memory capability, finds the global minimum in this complex landscape is still a mystery. The process is driven by forces of physics yet to be understood! Experiments suggest that a protein's approach to the global minimum is characterized by two phases. The first phase is a rapid folding phase that results in a nearly folded protein. This is followed by a lag phase which completes the folding process [50, 51]. This suggests the existence of a large energy barrier with many saddles around the valley containing the global minimum. See **Figure 3** [50].

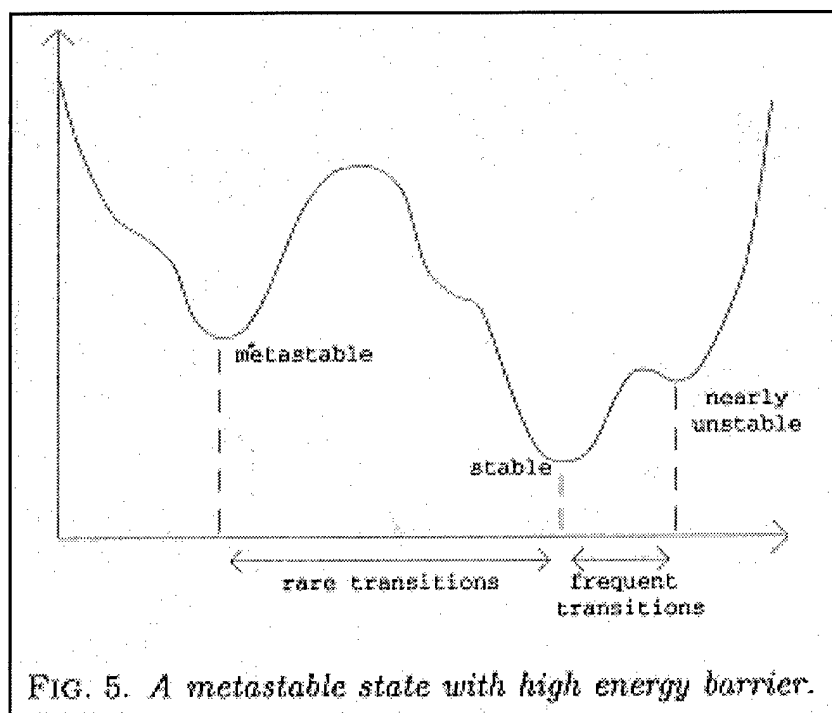


Figure 3: Metastable State

⁶ An invalid protein would be represented by extremely high conformation energy.

⁷ Some of the produced proteins cannot exist outside of this model.

Other scientists argue that the conformation state might be a metastable state with high barriers, or it might just be the lowest local minimum that is kinetically accessible from most of the protein's energy space [50]. These landscape descriptions allude to the possibility that a naturally occurring protein may not reach its global minimum energy conformation. This is supported by Kaiser's experiments that uncovered a conformation of [Met]-Enkephalin with a lower CHARMM energy value then previously encountered [52].

Figure 4: The Glassy Funnel

The most promising fitness landscape description is referred to as the glassy behavior [50]. (See **Figure 4** by [64].) The glassy behavior is defined by the situation when the naturally folded state corresponds to a more extended region in the search space where there are many closely located minima of approximately the same energy. The differences between the global minimum geometric structure and

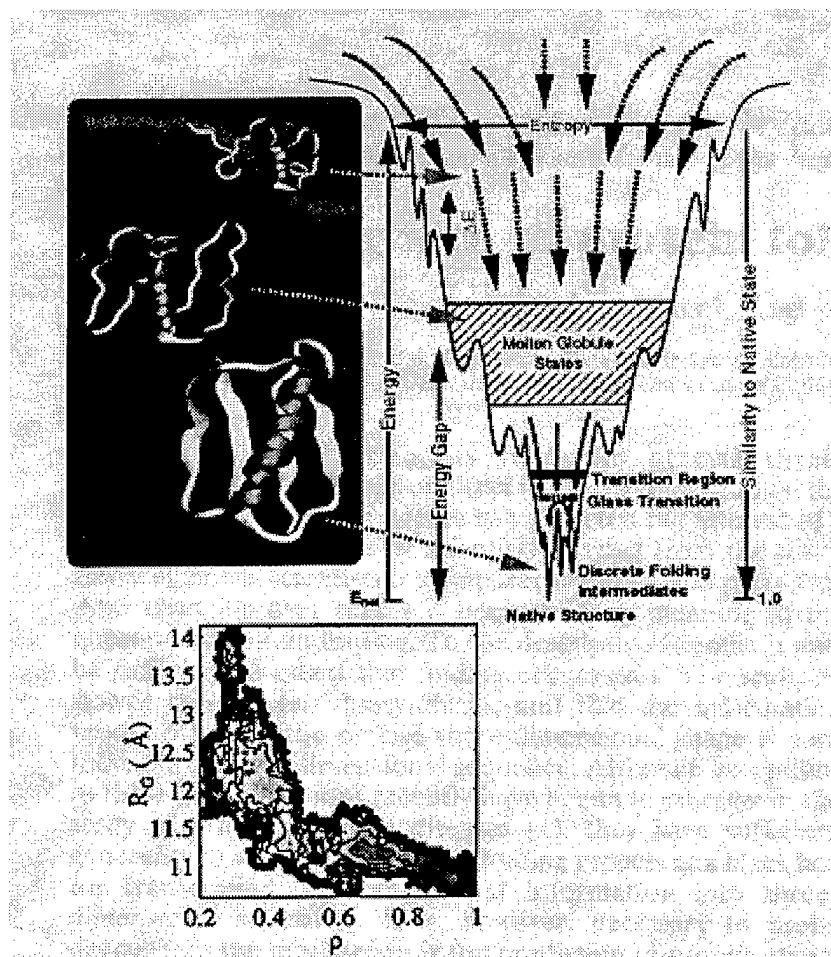


FIG. 1. The energy landscape for a folding protein. The major phenomenological parameters needed to capture this landscape include: the width of the funnel at small values of native similarity, indicating the entropy of denatured states; the roughness of the landscape, ΔE , which is related to the glass transition temperature, T_g ; the stability of the native state relative to the collapsed but non-native (molten) globule states, the energy gap. The ribbon diagrams of the α/β protein, segment B1 of streptococcal protein G (GB1) provide structures from ensembles of unfolded, molten globule, and native conformations. The folding landscape for GB1 is projected onto two coordinates, the radius of gyration, R_g , of the folding globule, and the fraction of native contacts, p , which indicates how close the folding protein is to the native. The free energy change as folding occurs is shown as a contoured surface: (native) state corresponds to the blue region and the most unfavorable unfolded state is represented by the green contours.

these false conformations are beyond our current scientific limits to measure⁸.

Furthermore, when one of these false conformations is entered into our chosen energy model, the resulting energy can differ from the naturally occurring conformation energy by only a few kilocalories. Normally, there is only 10 kcal/mol difference between the completely folded and unfolded conformation [37].

The glassy funnel landscape model combined with the experimental data of rapid initial folding followed by a lengthy lag time to reach the global minimum energy state explains the Levinthal paradox which has confounded researchers for years. The Levinthal paradox simply states that “the time a protein needs to fold is by far not large enough to explore even a tiny fraction of all the local minima believed to comprise the fitness landscape” [37, 50]. On the other hand, “when the slope towards the native conformation is dominant over the ruggedness of the landscape, folding kinetics is exponential and [therefore very] fast [50].” This insight allows us to picture the protein quickly folding to an orientation near the native conformation – i.e., the initial rapid folding period. Then, if we imagine this “near orientation” resting on a relatively smooth valley floor, “the lengthy lag period until the protein ‘finds’ its native conformation” can be conjectured as the protein searching this small area.

2.3 PSP Energy Models

In order to understand and manipulate proteins, we must be able to reliably predict the tertiary structure of the protein in a reasonable amount of time. Generally, there are three different methods to uncover the conformation state of a protein: X-ray Crystallography, Nuclear Magnetic Resonance, and Computational Models. X-ray crystallography and nuclear magnetic resonance spectroscopy are direct methods of measuring the position of each atom within a protein. These methods are extremely time consuming, error prone, and laborious⁹! Computational modeling, on the other hand, is somewhat less time consuming and easier to conduct, but these methods are approximations and may not precisely reflect the native structure of a particular protein. Although computational modeling has many shortcomings, it is still an area of utmost interest to biochemists because this form of calculating the native structure provides the greatest possibility of shortening the gap between the discovery or design of a new

⁸ Simulation time steps required to accurately model the folding process are on the order of a femtosecond (10^{-15} second) due to the thermal oscillations of bonded atoms [37].

⁹ See Appendix B. Current Methods for Protein Structure Prediction

protein and learning its conformational structure. **APPENDIX B** provides an overview of the x-ray crystallography and nuclear magnetic resonance spectroscopy, and an in-depth look at the different forms of computational modeling.

The particular computational model we are interested in is the CHARMM (Chemistry at **HAR**vard using **M**olecular **M**echanics) energy model. CHARMM, developed principally by Brooks and Bruccoleri [42], is an empirical energy function used in the investigation of the physical properties of a wide variety of molecules. The model is executed on a molecule at a particular temperature in a particular solvent (usually water). The model is based on separable internal coordinates and pairwise non-bonded interaction terms [42]. The model is a composite sum of several molecular mechanics equations. Each is decomposed into its terms in the following series of equations:

$$E_b = \sum_{bonds} k_b (r - r_0)^2$$

$$E_\Theta = \sum_{angles} k_\Theta (\Theta - \Theta_0)^2$$

Equation 5: Bond & Angle Energy Equations

Equation 5 accounts for bond and angle deformations which in most cases at ordinary temperatures and in the absence of chemical reactions are sufficiently small for the harmonic approximation to apply [42].

$$E_\phi = \sum_{dihedrals} |k_\phi| - k_\phi \cos(n_\phi)$$

Equation 6: Torsion Potential

The torsion energy term, **Equation 6**, is a four atom term based on the dihedral angle about the axis defined by the middle pair of atoms. For this term, the energy constant can be negative (indicating a maximum at the cis conformation¹⁰), and there may be several contributions with different k_ϕ and different periodicity's for a given set of four atoms [42].

$$E_w = \sum_{improper\,dihedrals} k_w (\omega - \omega_0)^2$$

Equation 7: Improper Torsion

¹⁰ See Figure 7.

The **Improper Torsion** term was developed to maintain chirality about a tetrahedral extended heavy atom¹¹, and to maintain planarity about certain planar atoms¹² with a quadratic distortion potential. Without this term, out-of-plane potentials tend to be quadratic. In addition, the term provides a better force field near the minimum energy geometry [42].

$$E_{LJP} = \sum_{\text{pairs } i \neq j} \left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^6} + \frac{q_i q_j}{\epsilon r_{ij}} \right]$$

Equation 8: Lennard-Jones Potential

The **Lennard-Jones Potential** equation accounts for the van der Waals forces of attraction and repulsion energy (the A_{ij} and B_{ij} terms) and the electrostatic attraction and repulsion energy [42]. This equation is the major contributor to the overall energy calculation.

$$E_{hb} = \sum \left[\frac{A'}{r_{AD}^{12}} - \frac{B'}{r_{AD}^6} \right] \cos^n(\Theta_{A-H-D}) \times \cos^n(\Theta_{AA-A-H})$$

Equation 9: Hydrogen Bonding Energy Reduction

Equation 9 accounts for a reduction in the van der Waals term between the hydrogen atom and the *acceptor* atom [42].

$$\begin{aligned} \text{Distance Constraints: } E_{cr} &= \sum K_i (r_i - r_{i0})^2 \\ \text{Dihedral Angle Constraints:} \\ E_{c\phi} &= \sum K_i (\phi_i - \phi_{i0})^2 \end{aligned}$$

Equation 10: Water-Water Interaction

The two equations, in **Equation 10**, account for water-water interaction when manipulating the solute in a water solvent. The distance “constraints” (atomic harmonic constraints) are used primarily to avoid large displacements of atoms when minimizing, while still allowing the structure to relax. The dihedral angle “constraints” are used to maintain certain local conformation or when a series of different conformations need to be examined in making potential energy maps [42].

$$E_{total} = E_b + E_{\Theta} + E_{\Phi} + E_w + E_{vdW} + E_{el} + E_{hb} + E_{cr} + E_{c\Phi}$$

Equation 11: Complete CHARMM Energy Equation

¹¹ E.g., an α carbon without an explicit hydrogen.

¹² E.g., such as a carbonyl carbon.

The CHARMM model is almost a verbatim implementation of **Equation 11**. The terms k_b and r_0 , k_Θ and Θ_0 , k_Φ and n_Φ , A_{ij} , and B_{ij} are empirical constants supplied as input. These parameters are calculated from “known” protein conformations supplied by the Brookhaven Protein Database (the official repository of protein structures) operated by the National Institute of Health. The number of bonded atoms, the number of atoms forming bond angles, atoms forming dihedral angles, and non-bonded atoms¹³ are determined based on the molecule supplied to the model and can be distinguished prior to model execution.

The AFIT implementation of the CHARMM model does not account for each of these terms. In the original implementation by Brinkman and the later revision by Gates, the hydrogen bonding reduction and water-water interaction terms are excluded because they do not significantly contribute to the overall molecular energy. Therefore, the AFIT implementation does not completely model the molecular interactions (see **Equation 12**), and we can imagine AFIT’s CHARMM implementation as modeling the molecular interactions in a vacuum. This is, of course, a common way to calculate the protein structure’s energy.

$$E_{total} = E_b + E_\Theta + E_\Phi + E_w + E_{LJp} + energyconst$$

Equation 12: AFIT's CHARMM Implementation

Furthermore, Gates indicated “other” errors in the primary implementation, and he corrected them in order to ensure AFIT’s model corresponded with the QUANTATM software package by the addition of the energy constant [14].

CHARMM was chosen as our energy function because it models the most contributing factors as compared to the other commercially available empirical energy function. **Table 2** provides a comparison between several currently available empirical energy functions of the energy terms they calculate. As the number of energy terms included within the model increases, the corresponding complexity/ruggedness of the protein energy landscape also tends to increase. The energy models listed in **Table 2** are in order by decreasing complexity of the energy landscape (e.g., CHARMM models the most complex landscape of those energy models listed). The smoother the landscape the less complex and time consuming it is to calculate the protein structure’s energy, because fewer terms are included. Of course, the calculated energy tends to be less accurate.

¹³ All atoms with three or more bonds separating them are considered non-bonded [15].

Initials	Name	E_b	E_θ	E_ϕ	E_w	$E_{\text{non-bonded}}$	E_{el}	E_{hb}	E_{cr}	$E_{\text{c}\phi}$
CHARMm	Chemistry at Harvard using Molecular Mechanics	X	X	X	X	X	X	X	X	X
AFIT CHARMm		X	X	X	X	X	X			
Amber	Assisted Model Building with Energy Refinement	X	X	X		X	X	X		
ECEPP/3				X		X	X	X		
OPLS	Optimized Potentials for Liquid Simulations			X		X	X			

Table 2: Comparison Between Commonly used Energy Functions

2.4 Bounding the Search Space

Kaiser's work greatly influenced our efforts at constraining the search space of the PSP problem. As we know, enumerating the whole (discretized) search space is an intractable problem. Therefore, if there were any "generic" way to limit the search space, it would be beneficial to incorporate into our algorithm.

Kaiser covers the basics of the geometry found within the backbone of a polypeptide and briefly discusses Ramachandran Plots [37]. But to fully understand Ramachandran's work, we must start by defining a peptide unit¹⁴. The peptide unit is a rigid planar array of four atoms: nitrogen, hydrogen, carbon, and oxygen [22]. The peptide unit is considered rigid and planar because the bond between the carbonyl carbon (referred to as either C_γ or C') and the nitrogen atom is not free to rotate. This bond has partial double-bond characteristics [22]. (See **Figure 5**.)

¹⁴ The peptide unit is the building block of all proteins. It is also commonly called an amino acid.

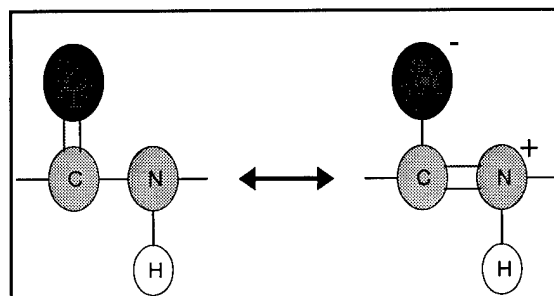


Figure 5: Partial Double-Bond Characteristics

Several peptides joined together by purely covalent bonding form a chain called a polypeptide [22]. The complete chain of peptide units define the backbone of a protein, and once a polypeptide backbone is configured with its appropriate side chain(s), it is commonly considered a protein¹⁵.

Rotations about the bonds within the protein are described as torsion or dihedral angles that are usually taken to lie between -180° and $+180^\circ$ [22, 63]. There are three distinct types of dihedral angles within the protein's backbone. **Table 3** lists how they are commonly referenced.

Using these conventions, a protein can be characterized as being in either the *trans* or *cis* position. The *trans* position refers to when each of the omega (ω) dihedral angles assumes a 180° orientation¹⁶ [22, 63]; on the other hand, the *cis* formation is characterized as the ω 's assuming an 0° orientation. The *trans* polypeptide form is naturally favored over the *cis* formation by approximately 1000:1, because, in the *cis* form, the C_α atoms and the side chains of the neighboring residue are in too close of proximity [63]. The closely positioned side chains greatly influence the pairwise atom interaction energy calculated by the repulsion term in the van der Waals equation (refer to **Equation 8** or **B.2.3.1.1.3 Non-Bonded Energy**). This term becomes very large when the distance between the atoms involved becomes slightly less than the sum of their contact radii [31]. **Figure 6** and **Figure 7** illustrate the *trans* and *cis* formations [63].

¹⁵ Proteins are produced in eukaryotic organisms through the process of transcription which begins with the transcribing of the DNA into messenger RNA which is then translated into a protein in the ribosome [65].

¹⁶ Protein angle numbering convention use a unit circle where 0° is at the top and $-180^\circ/+180^\circ$ is at the bottom. Negative degrees are measured counter-clockwise, whereas positive degrees are clockwise.

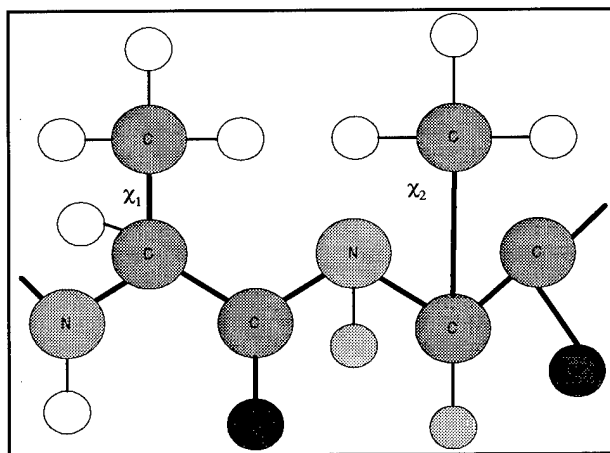


Figure 6: The *trans* Formation

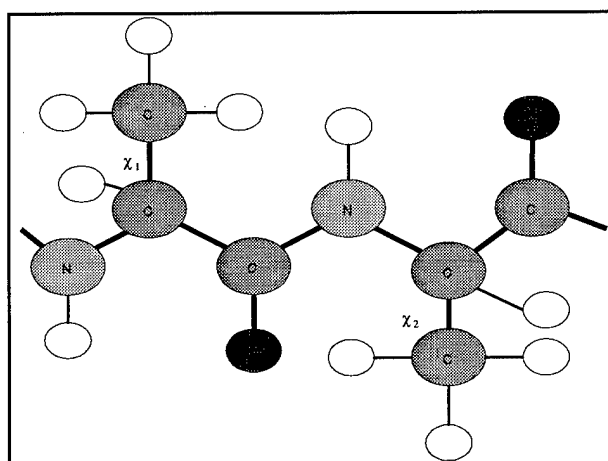


Figure 7: The *cis* Formation

Therefore, if we assume the ω dihedral angle is held in the *trans* position, then the phi (Φ) and psi (Ψ) dihedral angles define the backbone of a polypeptide [37]. Allowing slight deviations from the polypeptide's planarity of either the *trans* or *cis* conformation, by allowing the ω angle to deviate by -20° to $+10^\circ$, is thought to be only marginally unfavorable energetically in most peptide bonds [63]. Thereby, all three of the angles are responsible for defining the correct folded state of a large molecule.

Ramachandran *et al* developed constraints for allowable configurations for polypeptides based upon his two-parameter convention. Ramachandran proposed that it was possible to rotate around the N- α C and the α C-C γ when the groups were linked at the α C atom [61]. Consequently, the relative configuration of two peptide units about the α C atom are specified by just two parameters which he called ϕ and ϕ' [61]. (See **Table 3** for translation.)

Bond	Ramachandran	Standard
N- α C	ϕ	Φ (Phi)
α C-C'	ϕ'	Ψ (Psi)
C'-N	—	ω (Omega)

Table 3: Bond Angle Conventions

The complete configuration of a polypeptide chain is fully specified when each of the α C's parameters (ϕ , ϕ') are known [61]. Furthermore, Ramachandran developed a set of allowable regions for these parameters based upon his choice of permissible van der Waals contact distances using a hard sphere model of the atoms and fixed geometries of the bonds¹⁸ [61, 63]. **Table 4** has a comparison of permissible van der Waals distances as defined by Ramachandran and by Stryer [22, 63]. Ramachandran concluded that two sets of bounds were possible. These bounds, termed “normally allowed” and “outer limit,” were derived from a detailed analysis of available structural data including amino acids and peptides [61].

Contact	Ramachandran		Stryer Radii (Å)
	Normally Allowed (Å)	Outer Limit (Å)	
C ... C	3.20	3.00	4.0
C ... O	2.80	2.70	3.4
C ... N	2.90	2.80	3.5
C ... H	2.40	2.20	3.2
O ... O	2.80	2.70	2.8
O ... N	2.70	2.60	2.9
O ... H	2.40	2.20	2.6
N ... N	2.70	2.60	3.0
N ... H	2.40	2.20	2.7
H ... H	2.00	1.90	2.4

Table 4: Comparison of van der Waals Contact Distances

Based upon his steric constraints, “the permitted ranges for (ϕ , ϕ') were obtained, shown in **Figure 8** [61], corresponding to an angle of 110° between the N_0 - α C₁ and α C₁-C _{γ 1} at the α -carbon atom [61]. “When we allow this angle to vary slight from 105° to 115° , the allowed regions are altered slightly [61]”.

¹⁷ For the *trans* conformation, the range of ω is -160° to $+170^\circ$.

¹⁸ Commonly, refer to as Steric Constraints.

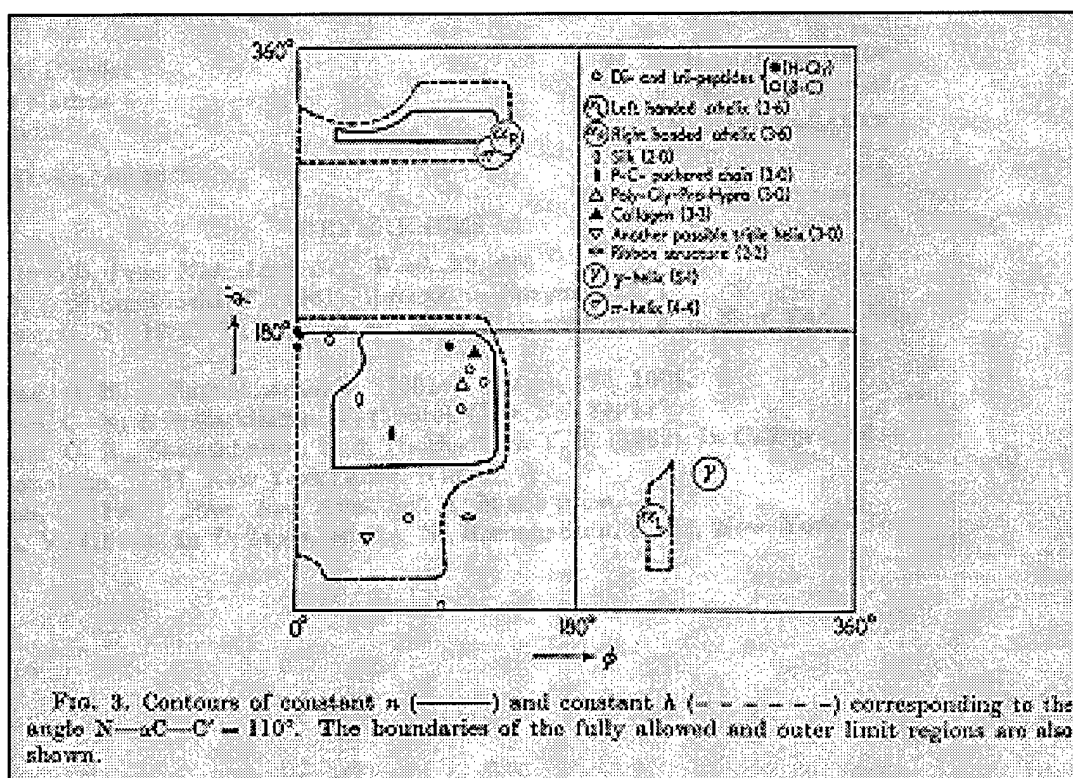


Figure 8: Original Ramachandran Plot

Our proposed constraint system is heavily based upon the work and results of Kaiser and the Ramachandran Plot. Kaiser's constrained-GA made use of an existing GA package, GENOCOP III. He incorporated his constraint system directly into the GA's manipulation of the chromosome [37], but his constrained-GA could not generate an initial population of 50 members using his defined feasible solution space because the feasible search space is much smaller than the entire search space [37, 52]. Therefore, Kaiser had to use a hand picked initial population.

It is commonly understood that any constraints placed upon a GA hampers its execution time. Normally, GAs have two choices when they encounter "disallowed" chromosomes: 1) excluded and replace¹⁹, or 2) repair the chromosome. If the disallowed chromosome is excluded and replaced, we may find that the GA spends an overwhelming amount of time finding "allowable" chromosomes, depending upon the ratio between the "allowable" search space and the "complete" search space. On the other hand, if we repair every "disallowed" chromosome, the GA must first recognize "why" the chromosome is not allowed and then repair the particular gene(s) in violation. This operation usually overwhelms the GA because it now must have problem domain

¹⁹ Kaiser's implementation followed this method.

information embedded within its algorithm. Summarizing Kaiser's work leads to the conclusion that constraints on the search space are "good," but his implementation lead to preconvergence and "islands" of feasible solutions that didn't allow his GA to traverse the search space to find the optimum solution [37].

What we propose is a better way to overcome the preconvergence and "islands" of feasible solutions situation. Our system guarantees that the chromosome encoding mechanism ensures that the allowable genes are represented and maintained throughout all GA operations.

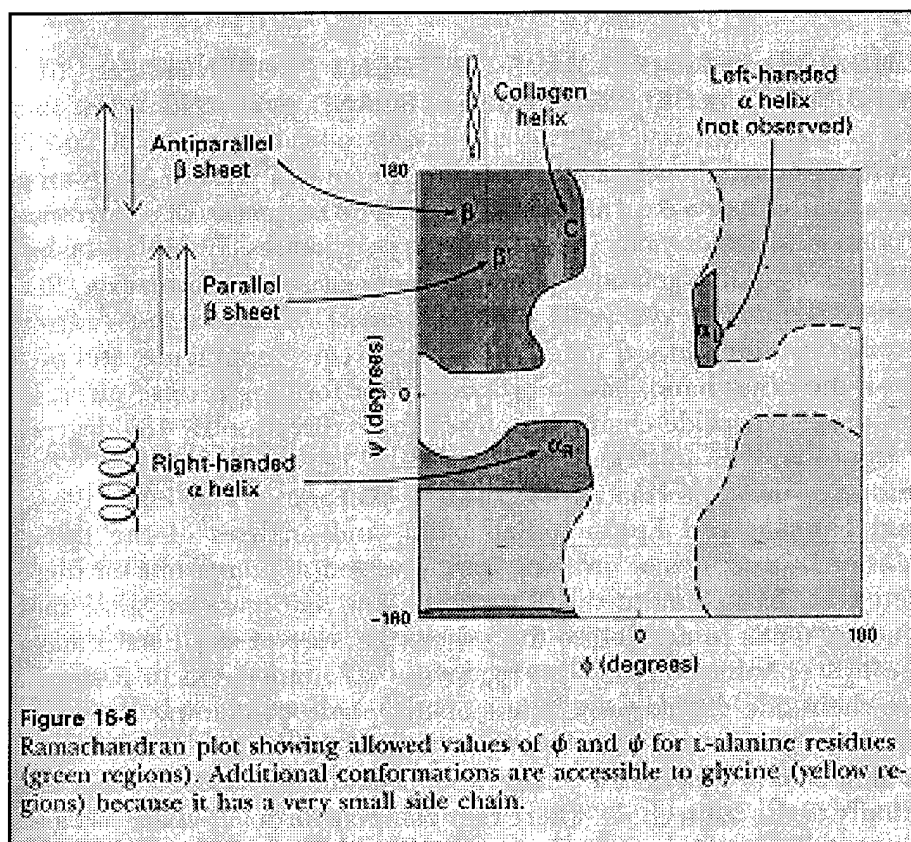


Figure 9: Stryer's Ramachandran Plot

We have devised another scheme to represent the search space using the Ramachandran Plots and affine transformations on the x- and y-axis that ensures all GA operators retain "feasible" solutions. The Ramachandran Plot is the key to our system! At first glance, **Figure 9** [22] makes it seem as if there are four distinct allowable regions based upon the values specified for (Φ, ψ) . But after a simple coordinate transformation, it is easy to see that the Ramachandran Plot doesn't actually create four

regions, but rather just one smaller region within the complete space illustrated in **Figure 10**. The transformation is mathematically defined as:

$$\begin{aligned}
 &\text{If } \Phi_{new} \leq 180^\circ \text{ then } \Phi_{normal} = \Phi_{new} \\
 &\text{If } \Phi_{new} > 180^\circ \text{ then} \\
 &\quad \Phi_{normal} = |\Phi_{new} - 360^\circ| \\
 &\text{If } \Psi_{new} \leq 180^\circ \text{ then } \Psi_{normal} = \Psi_{new} \\
 &\text{If } \Psi_{new} > 180^\circ \text{ then} \\
 &\quad \Psi_{normal} = |\Psi_{new} - 360^\circ|
 \end{aligned}$$

Equation 13: 1st Coordinate Transformation

Still there are “infeasible” regions within **Figure 10**. (i.e., The white space surrounding the yellow and green colored “bubble” represents unreachable Φ and ψ angles.) Therefore, we have relocated the coordinate system origin, (0,0), to correspond to two tangent lines and restricted the lengths of the axes to only span the feasible region - see **Figure 10** and **Figure 11**.

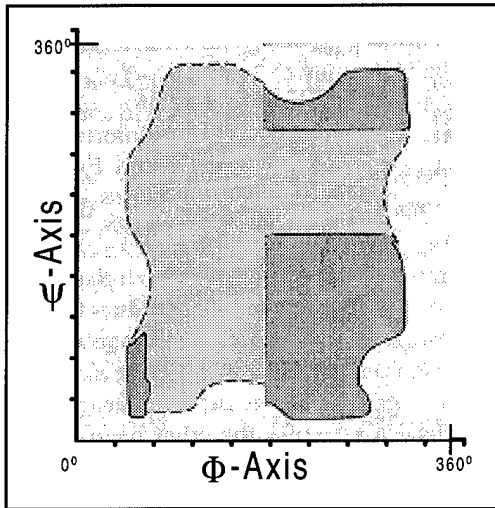


Figure 10: Applying 1st Transform

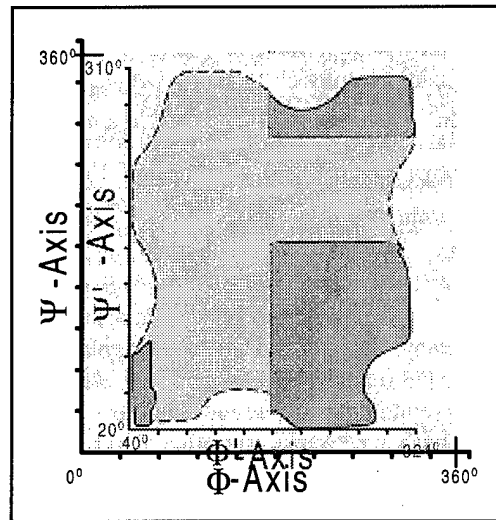


Figure 11: Repositioning the Origin

Now, the new Φ -axis (called Φ' -axis) corresponds to a tangent which intersects the point in the feasible region closet to the original Φ -axis, and likewise for the ψ -axis (called ψ' -axis). In past implementations, we have always assumed that the chromosomal encoding beginning at the origin (0°) and proceeding to 360° , represented by 2^0 to 2^{10} respectively. But with this new coordinate system, this is no longer the case!

The new (0,0) coordinate at the Φ' -axis/ ψ' -axis intersection is approximately 20° up the ψ -axis and 40° down the Φ -axis. Furthermore, the upper bounds of the Φ' -axis and ψ' -axis are less by approximately 36° and 50° , respectively. This second transformation is mathematically defined by:

$$\begin{aligned}\Phi_{new} &= \Phi' \left(\frac{(\Phi'_{upper} - \Phi'_{lower})}{360^\circ} \right) + \Phi'_{upper} \\ \Psi_{new} &= \Psi' \left(\frac{(\Psi'_{upper} - \Psi'_{lower})}{360^\circ} \right) + \Psi'_{upper}\end{aligned}$$

Equation 14: 2nd Coordinate Transformation

Figure 12 illustrates the 2nd transformation for the Φ -axis to the Φ' -axis.

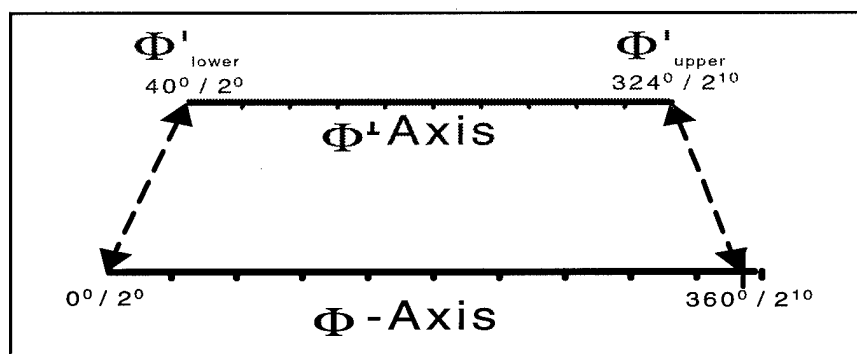


Figure 12: Φ -Axis Transformation

Our scheme allows the GA to process the chromosome's backbone independent of representation and guarantees that for all GA operators the transformed chromosome is in the defined "feasible" region of the Ramachandran Plot. These transformations only work for the polypeptide backbone configuration and does not account for side chains!

For the side chains, we consulted with Dr. Ruth Pachter (AFRL) to determine feasible ranges. Dr. Pachter validated the χ angles, as well as the backbone angles, proposed by Kaiser [37]. Therefore, the ranges he proposed for the backbone angles and side chains are used as our limits as well. This allows us to make direct comparison between his work and ours. Our constraint system incorporates the chromosome encoding explicitly (see **Table 5** and **Table 6** for our limits [15, 37]), and the transformations into the proper angular configurations are accomplished in the objective function (e.g., within AFIT's CHARMM energy model implementation). This ensures that all future AFIT PSP researchers can use any contribution from this

constraint system without having to manipulate their particular GA of choice. **Appendix I** contains Newman projections illustrating the constraints in **Table 5** and **Table 6**.

Dihedral	Midpoint	Radius	θ_{\min}	θ_{\max}
$\Phi_{\text{non-glycine}}$	-120	90	-210	-30
Φ_{glycine}	-180	135	-315	-45
Ψ	60	150	-90	210
ω	-180	20	-200	-160
χ_1	-60 60 180	30	-75 45 -185	-45 75 -165

Table 5: Loose Constraints for [Met]-Enkephalin

Dihedral	Midpoint	Radius	θ_{\min}	θ_{\max}
Φ	-67.5	22.5	-90	-45
Ψ	-30	30	-60	0
ω	180	20	-200	-160
χ_1	-60 60 180	30	-65 55 -185	-30 90 -150

Table 6: Loose Constraints for Polyalanine

2.5 Visualizing the Search Space

Visualizing the search space traversed by the GA is simply mind-boggling. If we assume the standard AFIT representation of 10 bits per dihedral angle and account for each dihedral angle in [Met]-Enkephalin on the x-axis, we yield a 240 bit representation for each chromosome which indicates one energy value on the y-axis. Since the x-axis is discretized, we can reduce this seemingly continuous line into a fixed interval using the natural numbers. Therefore, we have $1.7668\text{e}+72$ numbers across the bottom of a 2-dimensional graph. This number of x-values makes the problem of visualizing the GA's traversal through the energy landscape beyond the scope of available software tools and computational platforms. On the other hand, the requirement to understand this space remains. The purpose of this section is to explain how we intend to graphically visualize the PSP landscape.

Ideally, the best way to visualize the relationship between the molecule and the energy landscape would be in δ -dimensional space where δ represents the number of dihedral angles. This would reduce the problem to graphing 1024 points^{20} on each axis. Alas, there is also no mathematical way to represent 24-dimensional space. Therefore, we have derived a technique to reduce the numerical span of the first purposed visualization methodology to approximately 1934 points which is graphable using MathlabTM.

Mathematically, we have used a p-norm projection across the x-axis to make the discrete range of 2^{240} into a metric data scale using **Equation 15** where $p = 5$.

$$d(fn, fm) = \left[\int_{\Lambda} \left(|P_{FP}(D_n, \alpha) - P_{FP}(D_m, \alpha)|^p + |P_{TP}(D_n, \alpha) - P_{TP}(D_m, \alpha)|^p \right) d\alpha \right]^{1/p}$$

Equation 15: p-norm

Therefore, each 10-bit representation of a dihedral angle is a separate P_{FP} term within the equation. This transformation ensures that the representational distance between the x-axis values is maintained. Other norms were considered, but since we were trying to compress the data in order to produce a visualization which would easily fit on a single page without a reduction size the 5th-norm worked the best. **Figure 13** shows the compression rate of the a few different norms. The quicker the curve grows the more the data is compressed, but the representational distance between any two points on the x-axis is maintained!

²⁰ Each dihedral angle is represented by 2^{10} or 1024 discrete values.

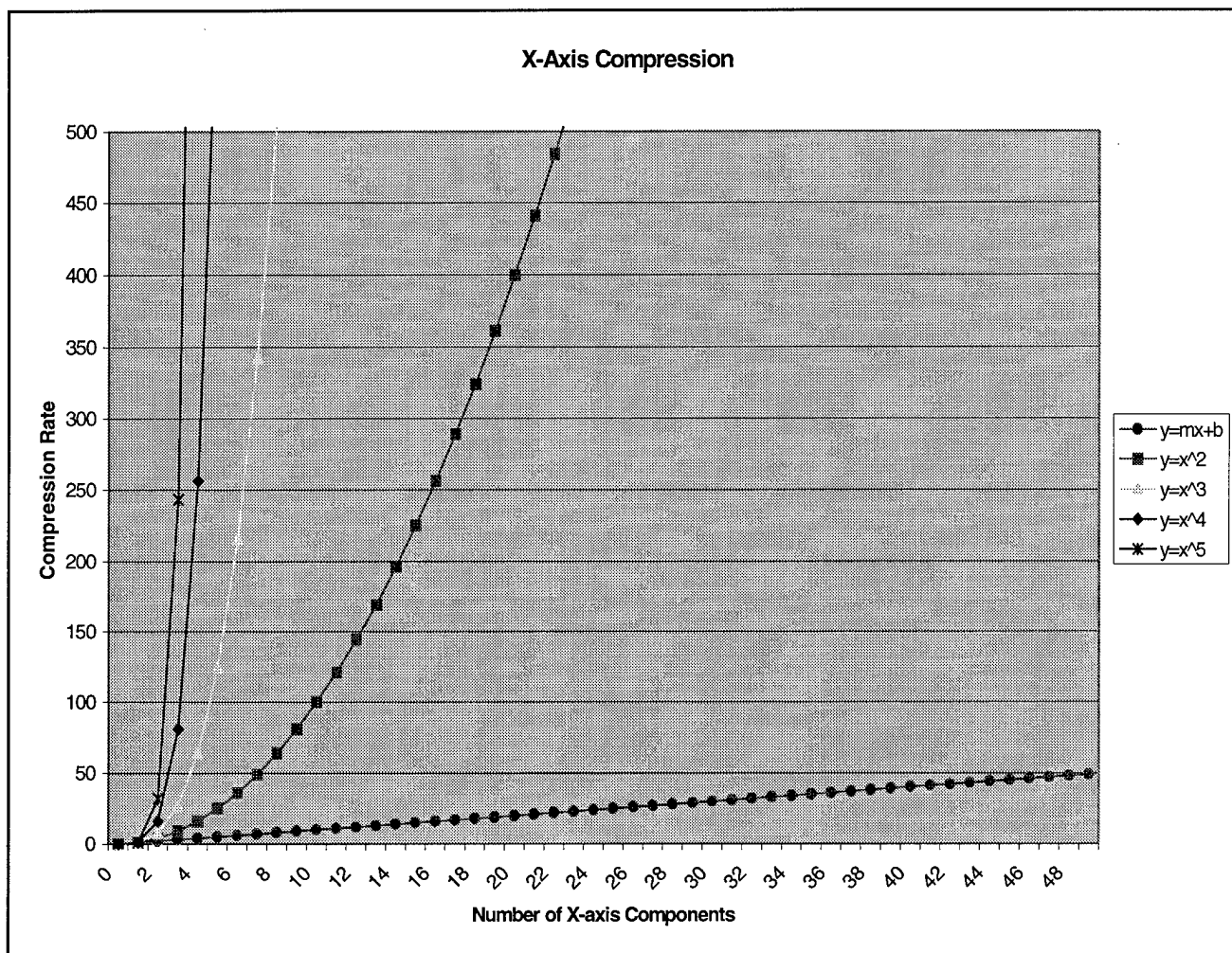


Figure 13: P-Norm Comparison

The y-axis, on the other hand, still represents the continuous real value range of the energy function. Since we are really interested in a small range of negative values and because it is not uncommon to have a $1e+32$ energy value associated with a molecule, we have chosen to bound the upper limits of the y-axis to +25 kilocalories. The remaining energy values are illustrated upon the graph by exploiting colored graphical gamut's located at zero and the x-axis intersection. **Table 7** indicates the color meaning and **Figure 14** shows some initial test data reduced as stated here.

Color	Range
BLACK (●)	$-\infty \rightarrow +25.0$
BLUE (x)	$+25.01 \rightarrow 1,000$
Yellow (▲)	$1,000.01 \rightarrow 1,000,000$
Red (◆)	$1,000,000.1 \rightarrow \infty$

Table 7: Visualization Legend

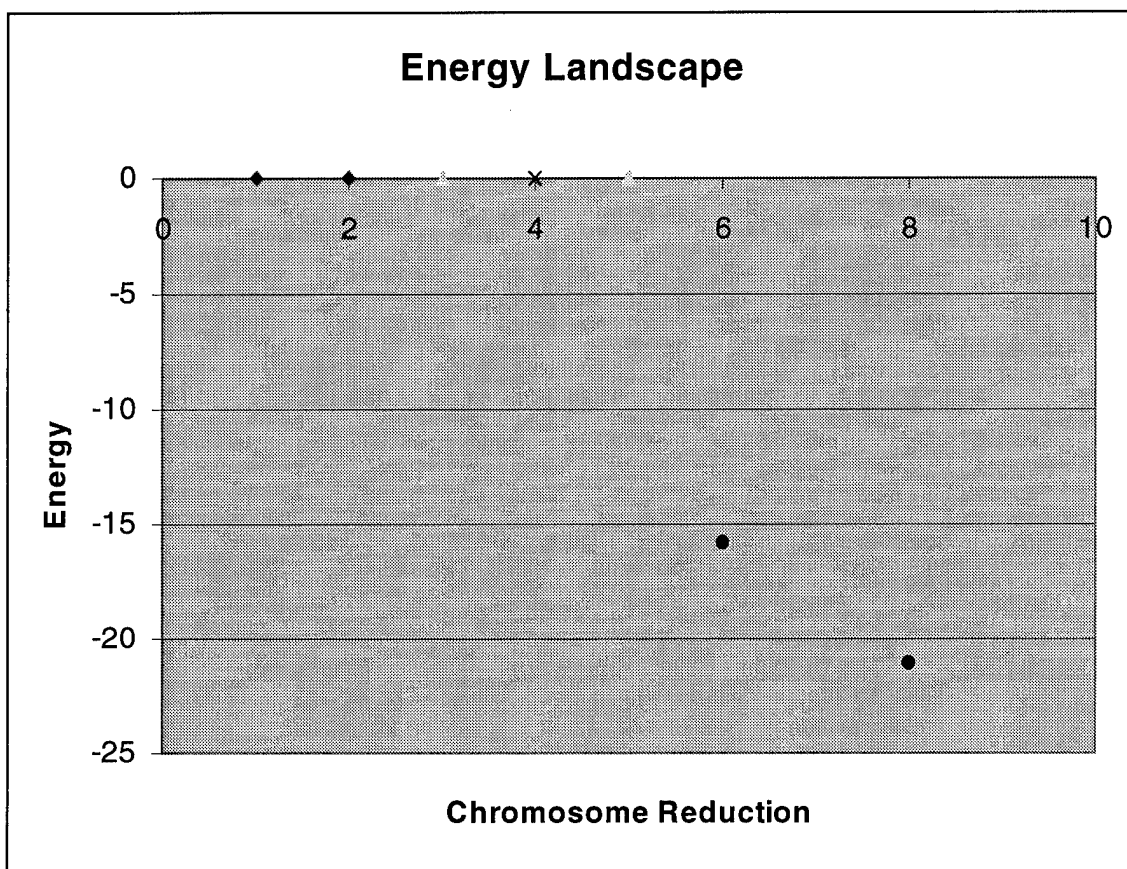


Figure 14: Energy Landscape Visualization

2.6 Summary

As this chapter indicates, the energy/search landscape of the PSP problem is huge, extremely complex, and poorly understood. Therefore, there can be enormous benefits reaped by constraining the space using accepted work from the PSP community, hence, our development of constraints. Kaiser [37] was on the right path when he developed the constraints on the dihedral angle, but his implementation left some modularity to be desired. Our new implementation restricts the search space just as effectively, may prove to be more efficient, and allows for a modular design by incorporation into the fitness function. Finally, our attempt to visualize the search space may lead to a greater understanding of the CHARMM energy landscape discretized by using 10 bits per dihedral angle. We could discover, however, that this discretization prohibits us from finding the global minimum conformation energy because our discretization scheme is too coarse. The next chapter discusses the different genetic

algorithms we investigated, and Chapter 4 provides implementation details for our constraint system.

3.0 Linkage Investigating Genetic Algorithms (LIGAs)

3.1 Introduction

Due to the simple genetic algorithm's (SGAs) inefficiency in applications involving a high degree of deception²¹, some genetic algorithm (GA) researchers conceived and gave birth to the family of linkage investigating genetic algorithms (LIGAs) [1, 2, 4, 5, 7, 8, 9]. The LIGA class of GAs explicitly emphasizes the importance and use of *building blocks*. Building blocks are schemata comprised of tightly linked genes. They consist of coupled values (locus/allele pairs) that work well together and tend to lead to improved performance when incorporated into a complete chromosome [13]. The biological concept reflecting "tightly linked genes" (i.e., the concept of linkage) refers to such bits acting as "co-adapted alleles" that tend to be inherited as a block (i.e., the building block) [6]. The *defining length* of a building block measures linkage. The defining length is the distance between the first and last bit of a building block, and it is a direct measure of how many crossover points fall within this significant portion the corresponding schema. This length determines the probability that the building block is disrupted during crossover [6]. As the defining length of a building block approaches the length of the chromosome, the probability of disruption increases because the crossover point probably occurs within the building block! The Schema Theorem, which implies that by passing on "good" schemata to the next generation increases the likelihood of finding better solutions, provides the symbolic foundation for searching and propagating building blocks with low order defining lengths [6].

In the remaining sections of this chapter, we explain and examine several different forms of genetic algorithms designed to uncover and propagate building blocks. This discussion of different types of LIGAs is not all encompassing nor is it intended to be complete because many linkage learning or building block propagating genetic algorithms have been proposed [1, 2, 4, 5, 7, 8, 9, 10, 11, 12].

3.2 Problems with the SGA

The LIGA class of GAs tries to combat two bottlenecks of the SGA proposed by Kargupta: 1) the combination of relation, class, and sample spaces, and 2) poor search

mechanisms for gene relations [9]. Kargupta explains that the first bottleneck derives from using a single population as the genetic pool. The *relation*, *class*, and *sample spaces* are combined with the decision making process. Therefore, each space affects the others in some undesirable way. The relation space defines “classes” in terms of the gene sequences within the chromosome. The class space equates to the building blocks found in the chromosome, and the sum total of all the chromosomes or the GA’s population is the sample space [9]. A real-world example would be “inbreeding.” In human populations where inbreeding is common, we find that the inhabitants demonstrate similar characteristics (i.e. the sample space). These characteristics are dominated by nearly identical DNA (i.e. class space elements) because the DNA is defined by a few nucleotide templates (i.e. relation space).

The other bottleneck can be contributed to the encoding of the typical SGA chromosome. Fixed-length and fixed-position genes characterize the SGA class of GAs. When the defining length of a relationship between genes grows large compared to the total length of the chromosome, the likelihood of disruption occurring during crossover grows exponentially. Therefore, the SGA is best suited for evaluating and processing only those relations that are defined over positions close to one another [9]. The family of LIGAs does not share this characteristic and as a result does not suffer from this bottleneck.

3.3 Survey of Current LIGAs

Almost all of the LIGAs discussed are based upon the *Schema Theorem* presented first by Holland in 1975 [6]. However, they only adequately address the single population SGA problem proposed by Kargupta [9]. The Schema Theorem simply states that “short, lower-order, above average schemata receive exponentially increasing trials in subsequent generations” [6] and three of the four LIGAs discussed follow this model explicitly. The other, the selfish gene algorithm [4, 5], follows the model implicitly, which is made evident in its discussion. The combination of relation, class, and sample spaces is partially handled by most LIGAs, but the complexity of this “evolutionary concept” is not yet well understood. The concept is partially based upon the meiosis and the production of gamete processes [10]. Therefore, until “we” can adequately explain how these operations execute in “real-world” evolutionary processes,

²¹ This class of problems is characterized by having coding function combinations that have misleading low-order building blocks which cause the GA to converge to sub-optimal points.

all attempts to model them in GAs will be only poor approximations of the complex natural process.

3.3.1 Messy Genetic Algorithm (mGA)

No discussion of the LIGA family of GAs would be complete without discussing the forerunner of all LIGAs: the messy GA (mGA). The mGA proposed by Goldberg *et al.*, in 1989, was a major paradigm shift for its time. The mGA was the first to suggest moving from “neat coding and operators” to allowing variable-length strings that may be under- or over-specified with respects to the problem being solved [1]. The original mGA was designed to handle the “deception problem,” but its usefulness is not limited to this realm. It is at least as efficient and effective as the simple genetic algorithm on both deceptive and non-deceptive problems in some test cases. Goldberg’s originally proposed mGA was fashioned from his view that nature’s climb out of the primordium occurred with genotypes that exhibit redundancy, over-specification, under-specification, changing length, and changing structure [1]²².

3.3.1.1 Chromosome Representation

Goldberg developed the mGA chromosome to allow for a relaxation in the coding of the gene by assigning each gene a “value,” called an allele, and a “location,” called a locus (e.g. {(allele, locus)}). Then, he took no steps to ensure that any particular chromosome contained a full complement of allele/loci pairs, nor to prevent redundant pairs, in accordance with his view of evolution. This led to two closely related problems. How to handle *over-specified* and *under-specified* genes within a chromosome. Over-specification occurs when a chromosome contains two genes that have the same locus value but differing allele values. The problem is: *what phenotype should the chromosome express when there are two competing gene alleles for a particular locus?* Goldberg’s solution to over-specification was to simply use *positional precedence* because of its simplicity. As the name suggests, positional precedence is based on a left-to-right scan of the gene with a first-come-first-served attitude when constructing a “complete” gene for a fitness evaluation. **Figure 15** illustrates how Goldberg envisioned the mGA’s positional precedence operation²³.

²² Appendix E.1 contains additional clarification of some mGA operations.

²³ Theoretical uses of the positional precedence operator is discussed in Appendix E.1a

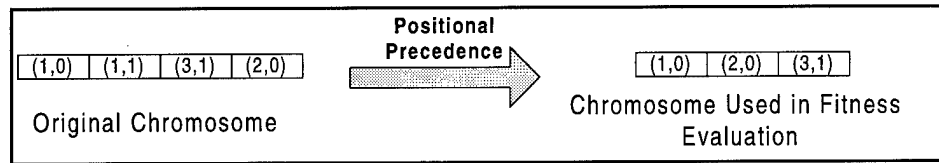


Figure 15: mGA Positional Precedence at Work

Under-specification is a much more difficult problem to overcome and is handled in a different and not so simplistic fashion. Goldberg originally assumed that the fitness function could be handled as a sum of non-overlapping subfunctions. This assumption allowed the mGA to evaluate every member of a population and compare them based on an “average” fitness [1]²⁴. This initial simplifying assumption proved not very useful nor scalable to “real-world” problems, and the handling of under-specified genes evolved into the use of a *competitive template* to “fill in the gaps” of the partially specified solutions. The competitive template method uses an *a priori defined “locally” optimal template* to fill in the missing bits of the partial solution. The locally optimal template is used to provide missing genes within the chromosome. This allows the fitness function to evaluate a completely specified chromosome²⁵.

3.3.1.2 mGA Algorithmic Phases

The mGA consists of three phases: *partially enumerative initialization*, *primordial phase*, and *juxtapositional phase*. In the partially enumerative initialization (PEI) phase, at least one copy of each possible building blocks of a specified size is created. These partial solutions make up the initial population²⁶, in contrast to random initialization found in most other forms of GAs. This phase is analogous to the predawn of life on earth when the sea was considered a “primordial soup” as first suggested by Russian scientist Alexander Oparin [67]. The primordial and juxtapositional phases can be thought of as two phases of selection.

In the primordial phase, the proportion of good building blocks is enriched through a number of generations undergoing reproduction without any other genetic operations. The objective is to create an enriched population of building blocks whose combination should create optimal or near optimal strings. Suboptimality of the final

²⁴ Each chromosome was evaluated by all possible fitness subfunctions then divided by the number of subfunctions to return an average fitness.

²⁵ Refer to Appendix E.1b for an in depth discussion of the competitive template.

²⁶ Appendix E.1c contains a complete description of PEI.

solution is possible, because the initial population instantiated by PEI does contain suboptimal building blocks. In some sense, we can think of this phase as a “weeding-out” of these suboptimal blocks. To meet this end goal, tournament selection is applied to the PEI generated population [15]. Tournament selection²⁷ is the only active operator during the primordial phase. Then, as selection proceeds, the population size is reduced by factor of two at regular intervals. This serves two purposes. First there is no need to maintain the population size associated with PEI once the better of the building blocks are chosen, and secondly the mGA reduces the population size in order for the population to be effectively and efficiently processed by the juxtapositional phase.

The juxtapositional phase resembles the usual processing of a SGA except the strings can vary in length. This phase proceeds with a fixed population size and the invocation of reproduction, *cut-and-splice* operators²⁸, and other genetic operators which can be included²⁹. Cut-and-splice was a novel contribution of the mGA to the realm of GA knowledge, and it acts to recombine the enriched proportions of building blocks passed on by the PEI and primordial phases. As long as the string lengths remain low, the action of cut-and-splice is likely to be as non-disruptive as simple crossover [1].

A high-level example of a mGA coding is provided in **Algorithm 1**:

²⁷ Tournament selection was used by Goldberg in his work, but any form of selection operator can be substituted.

²⁸ Appendix E.1d discusses this operator in terms of one-point crossover.

²⁹ In Goldberg’s original study, he used reproduction and cut-and-splice, but he eluded to the possibility of incorporating mutation.

```

Program mGA

begin
Do                                     /*outer loop*/
{
    PEI
    Evaluate Fitness of each Member of the Population
    Do                                 /*primordial phase*/
    {
        Selection(Tournament)
        Reproduction
        If (appropriate number of generations accomplished) Then
            Reduce Population Size
        End if
    } until the maximum number of Primordial Phase are
accomplished
    Do                                 /*juxtapositional phase*/
    {
        Cut_And_Splice Operator
        Other GA Operations
        Evaluate Fitness of each Member of the Population
        Selection(Tournament)
    } until some stopping condition is met
    Save best solution as next kth-order iteration's template
    } until problem domain's order of deception is accomplished
    Save "best" solution found as finally output
end;

```

Algorithm 1: mGA Pseudo-Code

3.4 Selfish Gene Genetic Algorithm (SG GA)

The SG GA proposed by Corno, Reorda and Squillero (1998) follows a somewhat nontraditional view of evolution. A "traditional" GA follows the evolutionary views proposed under Darwinism. Their common underlying assumption is the existence of a population of individuals that strive for survival and reproduction [4]. The basic unit of evolution in these traditional algorithms is the individual, and their goal is to find an individual of maximal fitness [4]. On the other hand, the SG GA follows a recently proposed view of evolution where the fundamental unit of natural selection is the gene rather than the individual. The selfish gene theory of evolution, proposed by Richard Dawkins in 1976, claims that whereas the individual eventually does not survive, but the genome of the individual is able to replicate itself into subsequent generations potentially indefinitely [4]. In the population, the important aspect is not the fitness of

each individual, since those individuals are mortal. (e.g., their “good” qualities are lost with their deaths [4].) For instance, in the ideal case a child of a diploid organism receives half of the genes from one parent and half from the other. Therefore, a grandchild only represents a fourth of each grandparent’s genes, and so on (see **Figure 16**).

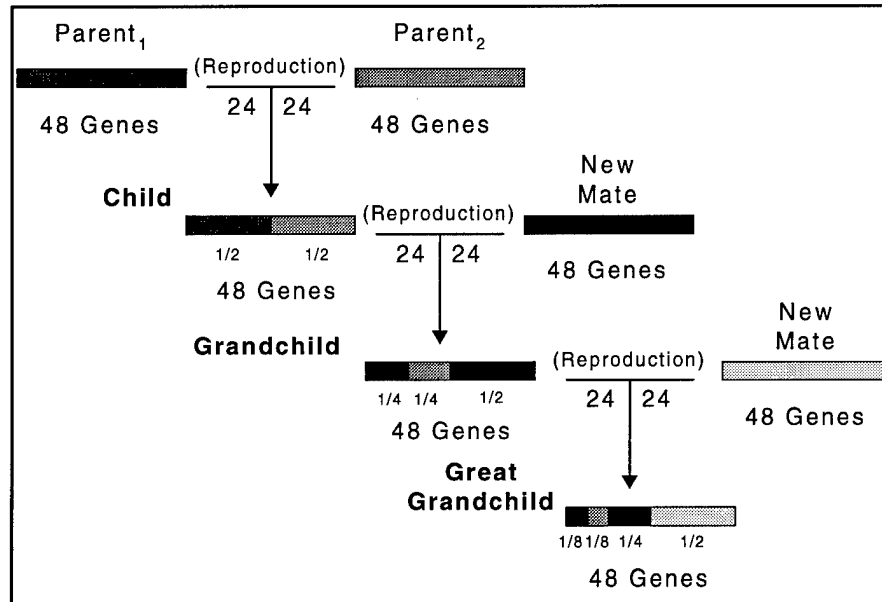


Figure 16: Propagation of a Chromosome Fragment

Individuals, therefore, are viewed as fleeting in the sense that their “life” in evolutionary terms is nearly spontaneous because the evolutionary process takes eons. On the other hand, genes live forever in the sense that a fragment of a chromosome survives the individual and is replicated in its offspring: the gene survives the death of the individual. In the selfish gene concept of evolution, individual genes strive for appearance in the genotype of the individuals, whereas the individual is nothing more than a vehicle allowing the genes to reproduce. Due to the shuffling of genes that takes place during sexual reproduction, “good” genes (i.e., good building blocks) are viewed as those genes that, when combined with other genes, give higher reproduction probabilities to the offspring [4]. For example, if a gene is able to produce a useful characteristic, then the individual with that characteristic (gene) in their genome has a higher probability of breeding. Thus, such genes have a higher probability to spread in the gene pool and therefore receive greater representation in future generations³⁰.

³⁰ Appendix E.2 begins the additional coverage of the SG GA.

3.4.1 SG GA Chromosome Representation

Since the SG GA does not maintain an instantiated population of individuals, the SG GA relies upon a *virtual population*. The virtual population is an abstract model aimed at representing the *gene pool* concept defined by Dawkins. The gene pool is a collection of all possible allele values for each locus position in the genome. As in the mGA, each gene is given a value and a position.

Since individuals do not persist in the SG algorithm, and therefore “fitness” is not associated with any particular set of genes, the SG algorithm models reproduction through its effects on *statistical parameters* that model the virtual gene pool. The statistical parameters model the virtual population at two levels. First, the success of a particular allele is measured by the frequency with which it appears in the virtual population. Since any locus can take-on any one of several alleles, the probability of expression of each allele, independent of the alleles found in other loci, is stored as a component of a *marginal probability vector* (MPV) for each locus (L), (i.e., the $MPV_{L1} = \{a_1, a_2, a_3, \text{etc}\}$). The marginal probability vector is a collection of frequencies for each value an allele can assume. At the next level, the virtual population is statistically characterized by the collection of marginal probability vectors (MPV_{total}) for the various loci. The collection is stored in a single array where the length of the array equates to the number of loci and at each cell of the array there is the marginal probability vector for that particular locus (i.e., $MPV_{\text{total}} = \{MPV_{L1}, MPV_{L2}, MPV_{L3}, \text{etc}\}$). It is important to note that the MPV_{total} is not required to be square because each locus is allowed to have its own allelic alphabet.

3.4.2 SG GA Algorithmic Phases

The SG GA follows two steps in its processing: 1) *initialize* gene pool and 2) *reproduction* based on fitness and tournament selection. Initialization of the gene pool is motivated by the principles outlined above in the discussion of the virtual population. All possible genes are created and the marginal statistical probability vectors are calculated for the complete virtual gene pool³¹. The virtual gene pool starts with all alleles for each locus having an equal probability of expression. The probability of expression for each allele evolves through the process of reproduction.

³¹ Appendix E.2a investigates the growth rate of the virtual gene pool as the length of a chromosome increases.

The process of reproduction is discussed as three phases: *generation of individuals*, *tournament selection*, and *replication*. An individual is created/formed only when needed for competing in a tournament, and then it is immediately discarded [4]. Two individuals are created from the virtual gene pool. For each locus in each chromosome, the allele chosen for the representation in the individual is either selected by mutation or based on the MPV. If mutation is warranted, a random allele chosen uniformly from the locus' allelic alphabet set is used. Mutation is modeled by random occurrence with a very low probability (P_m)³².

If $random_number(0,1) < P_m$ then chose
random allele

Equation 16: Selfish Gene Model of Mutation

Next, these two individuals undergo tournament selection based upon their phenotypical characteristics, and the one with the higher fitness is considered the winner. Finally, all alleles appearing in the winning individual/chromosome slightly increase their probability of expression in their respective loci in the virtual population; the losing chromosome's alleles are proportionally decreased in the corresponding loci. The allele frequencies are increased/decreased by some predetermined constant (ϵ)³³.

This form of replication is not considered asexual reproduction because of reshuffling the genes creates a blind cooperation between genes in the winning "individual." The rewarded alleles are selected together with other alleles, in other loci, different from the ones appearing in the former winner [5], therefore this new winner is not an identical copy of the former winner as would be the case in asexual reproduction.

This process continues until some stopping condition is reached. Corno's initial SG GA stopped when the genetic algorithm reached a *steady state*. The SG GA defines a steady state exist, for each locus l , of an allele (a_{lv}) whose probability of expression is over a given threshold p_l ³⁴ [4, 5]. Mathematically, the steady state is defined as:

$$\forall l : \max_v(a_{lv}) > p_l$$

Equation 17: SG Steady State

³² Appendix E.2b discusses SG GA mutation in terms of convergence rate.

³³ Appendix E.2c analyzes this epsilon feedback loop.

³⁴ p_l values are usually around 0.95

When this condition is met, all individuals modeled by the virtual population are very similar. In fact, if not for random mutation they would be identical. Therefore, the virtual population is not likely to evolve any more [4].

A high-level example of a SG GA code is provided in **Algorithm 2**:

```

Program SG

begin
  initialization
  Do {
    Select individuals
    Determine fitness of each individual
    If (fitness of individual1) < (fitness of individual2)
      Reward alleles (individual1)
      Penalize alleles (individual2)
    else
      Reward alleles (individual2)
      Penalize alleles (individual1)
    End if
    Discard individuals
  } while (stopping condition is not reached)
end;

```

Algorithm 2: SG GA Psuedo-Code

3.5 Gene Expression Messy Genetic Algorithm (GEMGA)

GEMGA³⁵ is another compelling investigation into the linkages between genes as proposed by Kargupta in 1996. GEMGA's foundation is rooted in an alternate perspective of blackbox optimization (BBO) in terms of relations, classes, and partial ordering which Kargupta coins as SEARCH (Search Envisioned As Relation and Class Hierarchizing) [9, 10, 11, 45]. SEARCH is motivated by the observation that searching for optimal solutions in BBO is essentially an inductive process and in absence of any relation among the members of the search space, induction is no better than enumeration [11]. SEARCH (a complete description can be found in [9]) decomposes BBO into three spaces: 1) relation, 2) class, and 3) sample space [9, 11]. Relations divide the sample space into different classes. The sample space is the area we are searching, and the classes delineated by the relations can be viewed as a pruning mechanism. If the algorithm divides the sample space into relations, we can order the classes based upon "better" relations and organize the members of each class based

³⁵ Appendix E.3 begins the additional discussion of GEMGA.

upon their “goodness contribution” to the relation. In this manner, we have effectively pruned the search space by discounting the “lesser” classes because they represent worse relations in the sample space. There are five major components to SEARCH (see **Figure 17**), and GEMGA is a distributed implementation of each of these steps.

- Classification of the search space using relation
- Sampling
- Evaluation, ordering, and selection of better classes
- Evaluation, ordering, and selection of better relations
- Resolution

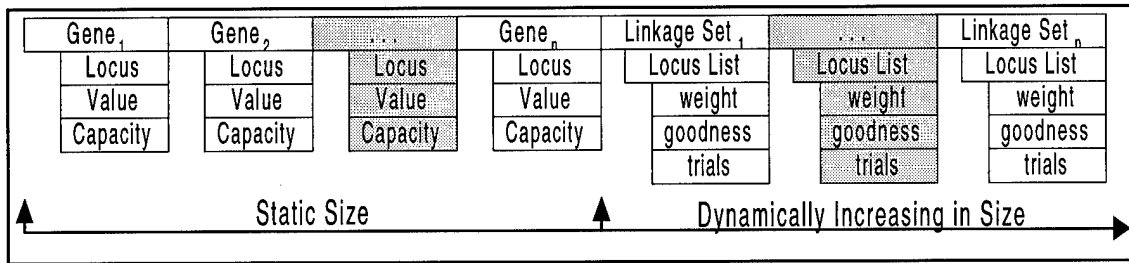
Figure 17: Steps of SEARCH

3.5.1 Chromosome Representation

According to the originally proposed definition of “messy” by Goldberg, GEMGA is not messy at all. GEMGA does not allow under- or over-specification. Instead, it follows the more traditional views of a fixed length fully specified chromosome. The chromosome representation found in GEMGA is fixed length string where each member (gene) is a complex data type similar to the one presented in the mGA. Each gene representation in GEMGA contains three values: the *locus*, *allele*, and *weight* [9]³⁶.

Besides genes, the chromosome also contains a dynamic list of lists called the linkage set [10]. The linkage set replaced the gene characteristic of the linkage set found in earlier versions of GEMGA. The purpose of the linkage set is to define a set of genes that are related for each locus. The linkage set is actually comprised of a list of weighted lists, called locuslist. Each locuslist contains three related factors: the weight, goodness, and trials. The weight measures the number of times that the genes in the locuslist are found to be related in the population, whereas, the goodness relates how strong the linkage of the genes are in terms of their contribution to the overall fitness of the chromosome. Finally, the trial field indicates the number of times this linkage set has been tried [10]. The whole gene representation and linkage set dialogue defines the relation space of the GEMGA population. (See **Figure 18**)

³⁶ Appendix E.3a explains each of these gene characteristics.



° **Figure 18: GEMGA Chromosome Representation**

3.5.2 Algorithmic Phases

The GEMGA algorithm has three phases: Initialization, Transcription stage (formerly Primordial Phase [9, 11, and 45]), and RecombinationExpression stage (formerly Juxtapositional Phase [9, 11, and 45]) [10]. Note that in the GEMGA documentation, Kargupta specifies that GEMGA only has two stages, Transcription, and RecombinationExpression. He assumes that the algorithm's population has already been initialized [10]. During initialization, GEMGA creates the initial random population of chromosomes under the requirement that at least 1 instance of the optimal order-k class must be in the population³⁷. In order for the population to contain at least a single occurrence of the order-k member in the population (n) there must be $c|\Lambda|^k$ members where c is some constant that depends upon the variation of fitness values of the members of schema [10], and $|\Lambda|$ is the cardinality of the alphabet. Since in practice the order of delineability is unknown, Kargupta suggests that the choice of a population's size determines what order-k relationship GEMGA should process [10]. Therefore, after some algebraic manipulation, he presents the following equation:³⁸

$$k = \frac{\log\left(\frac{n}{c}\right)}{\log|\Lambda|}$$

Equation 18: GEMGA Population Requirement

During the Transcription stage, the *transcription operator* is applied deterministically for all ℓ genes in every chromosome of the population for ℓ generations [10]. The transcription operator applies a random subset of all alphabet transformations

³⁷ Order-k represents the complexity of the linkage GEMGA is investigating.

³⁸ Appendix E.3b explores the population requirements encouraged by this equation.

to every gene one at a time³⁹. The value of the gene is flipped to a different element of the allelic alphabet and the change in the fitness value is noted. For example, if a chromosome of length 4 were encoded using a binary representation, then in the first generation the first bit would be flipped for every member of the population. Next, each chromosome would undergo a fitness evaluation to determine if the particular bit flip improved the overall fitness of the chromosome.

If the chromosome's fitness improves as a result of one of these changes, then the original chromosome is not likely to be a member of the optimal schema defined over a partition that subsumes the gene under observation. On the other hand, if the fitness worsens, then perhaps the gene belongs to a good class – i.e., it has strong linkage⁴⁰.

Once the chromosome's fitness is evaluated, the capacity of that gene is set to either 1 (the gene has a capacity to change) or 0 (the gene has no capacity to change). The choice depends on whether the mutation of the allele value had a positive or negative effect of the chromosome's overall fitness.

Once all the alleles have been examined, those genes whose capacity changed to zero are collected and stored in the first element of the linkage set for each chromosome. These genes are called the initial linkage set. **The transcription operator only changes each gene's capacity and initiates the formation of the chromosome's linkage sets.** At the end of this stage, the chromosome has its initial fitness and configuration restored [10].

Once the transcription phase is complete, the RecombinationExpression phase begins with the "modified" population. The RecombinationExpression stage is actually two separate subphases: PreRecombinationExpression and RecombinationExpression. The RecombinationExpression phase continually applies these two subphase until some predefined stopping condition is met⁴¹. A high-level example of a GEMGA genetic algorithm coding is provided in **Algorithm 3**:

³⁹ In the case of a small allelic alphabet, it is assumed GEMGA progresses through all possible allelic values.

⁴⁰ The scenario is for a minimization problem -- reverse for a maximizing optimization problem.

Program GEMGA

```
begin
  Initialization                      /* Initialize Random Population */
  j = 0
  Do                                  /* Find Better Relations */
  {
    Apply Transcription Operator
  } Until (j == problem - length)
  for (i=0, i <= Numberoftrials, i++) /* Define Relations Between Genes */
    Apply PreRecombinationExpression Operator
  Do                                  /* Selection and Crossover */
  {
    Apply GEMGA Recombination
  } Until (some stopping condition has not been meet)
end;
```

Algorithm 3: GEMGA Pseudo-Code

3.6 Linkage Learning Genetic Algorithm (LLGA)

The LLGA is another attempt by Goldberg and his students to find/create a *competent* GA. Goldberg defines a competent GA as one that “can solve problems of bounded difficulty quickly, reliably, and accurately [8].” The LLGA was first proposed as a new linkage-investigating algorithm by Harik in 1996. Harik argues that other implementations of GAs do not take explicit advantage of “tight linkages” early enough in their algorithmic processing. If they did (as does the LLGA), then they would be able to solve “difficult problems [7].” The LLGA takes advantage of tight linkages between genes by using a new two-point crossover operator and a different chromosome representation.

In order to understand the LLGA, we must comprehend Harik’s *et al.*’s new definition of building block linkage that **only** applies to this genetic algorithm. According to the LLGA research, building block linkage is defined as the *probability* that building blocks are conserved under whichever crossover operator is used [1]. This definition contrasts with the popular view of “building block linkage” present by Whitley [6] and earlier Goldberg papers [1, 2]. Whitley and Goldberg equate linkage with physical adjacency on a string as measured by defining length, and defining length is based on the distance between the first and last bits in the schema [6].

⁴¹ Appendix E.3c defines the RecombinationExpression phase.

Normally, the GA community considers building block linkage to mean the definition supplied by Whitely (and others), and we will refer to that definition as the classical building block linkage definition while discussing the LLGA to avoid confusion. Harik, on the other hand, argues that although this definition is appropriate for one-point crossover, it is imprecise for other crossover operators such as uniform and two-point crossover [17]. His new crossover operator is a variant of two-point crossover, and thus, he uses two-point crossover as a means to explain his building block linkage definition.

We can picture two-point crossover operator as treating the chromosome as a string of beads connected in a circular list. If we imagine a fixed circumference necklace as the number of beads within the necklace grows to infinity, the thickness of each individual bead drops to zero [7, 17]. Consequently, we can view the circle of beads as having a circumference equal to one. Harik defines a k-order building block as having k points on this circle, and he labels the successive distances between these k-points as y_1 through y_n . Therefore, a building block is preserved under two-point crossover precisely when the injected genetic material falls within one of these gaps [17]. He suggests an equation to calculate his new building block linkage as *“the probability of both crossover points falling within the same gap equals the sum of the squares of the gap lengths [17].”* (See **Equation 19**)

$$\lambda = \sum_{i=1}^n y_i^2$$

Equation 19 : LLGA Building Block Linkage

Figure 19 graphically shows how Harik interprets building block linkage [46].

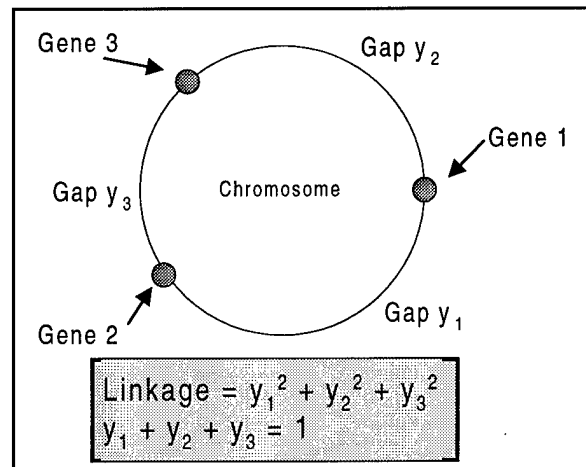


Figure 19: Harik's Linkage Definition

3.6.1 LLGA Chromosome Representation

In Harik's version of the chromosome, each gene has a value and a position (allele/locus pair), but the chromosome is not allowed to demonstrate under-specification as in the mGA. Over-specification, on the other hand, is always present. Each chromosome is completely over-specified⁴². The allele chosen for expression is based upon positional precedence. In the original usage, the "positional precedence" operator (refer to Goldberg's mGA) was defined as meaning that a "complete" chromosome is constructed by a simple left-to-right scan of the linear chromosome and the first allele value for a particular loci encountered is expressed during the fitness evaluation. (See **Figure 15**.)

In contrast, Harik's views the chromosome as a circular list of genes. (See **Figure 20**.) Somewhere along this circular list is an *interpretation point*. The interpretation point serves as the starting location from which the fitness evaluation function begins to interpret the genes of the chromosome in a clockwise manner recording the first occurrence of each gene as the expressed characteristic. Starting from the interpretation point, Harik's version of positional precedence operator functions exactly the same as Goldberg's originally envisioned positional precedence operator. The difference is that the location of an individual's interpretation point changes during the LLGA's processing in order to allow for other allelic expressions. In this manner of

⁴² Harik's work assumes a binary encoding allelic alphabet, but if we allowed for some other alphabet, we are required to represent each allele value in the chromosome for each locus

encoding and interpretation, diversity is never lost because the chromosome contains every allelic value.

Furthermore, within the chromosome, Harik includes *non-coding* material called introns. The introns give no contribution towards the fitness of an individual and are not included in the chromosome's expression, but serve to facilitate the propagation of building blocks and the formation of linkage⁴³.

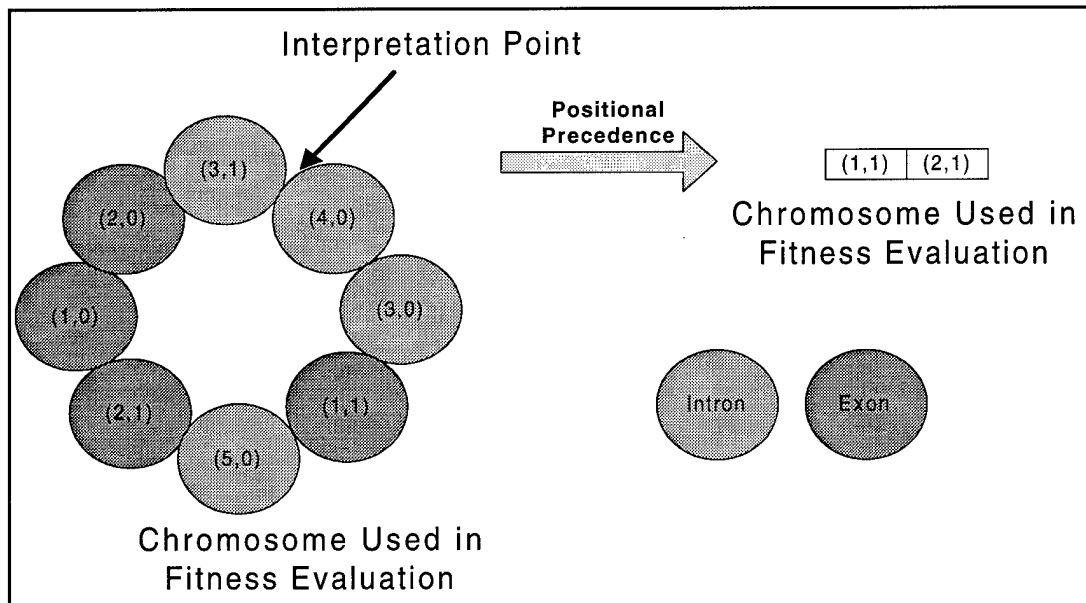


Figure 20: Visualization of a Chromosome

3.5.2 LLGA Algorithmic Phases

The LLGA executes in a similar fashion as the SGA⁴⁴. Specifically, the population is initialized, and then selection and crossover are applied generation after generation until some stopping condition is met. Any form of selection may be employed, but tournament selection with its low rate of convergence due to its minimal selective pressure allows the LLGA more “time” to explore/uncover linkages [7]. Harik coins a new crossover operator for the LLGA, which he calls the *exchange operator* [7, 8]. This operator requires two chromosomes selected from the population for reproduction. One of the chromosomes is designated the *donor* and the other the *recipient*. The operator selects a random segment of genetic material from the donor and grafts it into the recipient at a random location. Since both chromosomes are assumed to have an implicit orientation, the grafted alleles/loci are in the same

⁴³ The function and number of introns necessary is discussed in Appendix E.4b

⁴⁴ Additional discussion of the LLGA begins in Appendix E.4

orientation as they were before this procedure. Now, the recipient is considered “overfull” because it has duplicate copies of various allele/locus pairs.

The duplicate introns/exons pairs are deleted according to the following protocol. First, the interpretation point is transferred to the location of the first gene grafted into the recipient. Then, starting at the new interpretation point and going in a clockwise manner the genes are recorded. When a duplicate gene is found before completing the circle, it is deleted. In this manner, the genes transferred from the donor remain intact in the recipient⁴⁵.

The genes in the recipient are brought closer together by two subtle mechanisms operating within the exchange operator. First, the genes that survived crossover in the recipient are brought closer together by the deletion process. Duplicate introns and exons are pruned from the original chromosome. This results in those remaining genes having a smaller/tighter defining length (see **Figure 21**) as defined by Whitley and a lower building block linkage probability as defined by Harik (e.g., classical $5 \rightarrow 3$, new $2 \rightarrow 0$ from **Figure 21**).

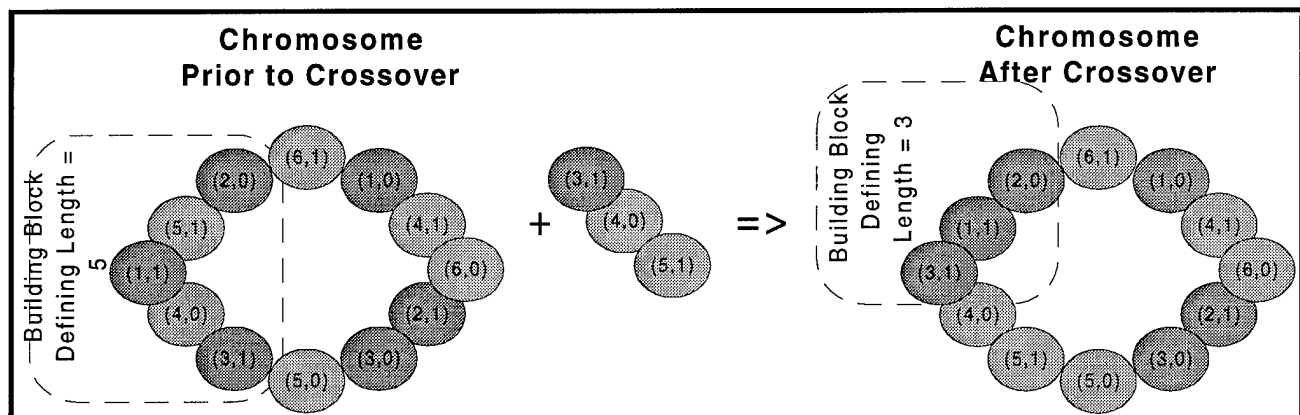


Figure 21: Deletion Process Tightening of Building Blocks

The second mechanism radically changes the defining length of a “good” building block using either definition. For instance, suppose we have the following building block comprised of (1,1), (2,0), and (4,1) with a building block linkage of (classical = 12) and (new = 10). Prior to crossover the chromosome resembles **Figure 22 Step 1**. During crossover, the donor injects the following genetic material {(9,0), (1,0), (5,0), (4,1)} in front of the first element of the building block in question, and this new material contains an element of the building block {gene (4,1)} (**Figure 22 Step 2**).

⁴⁵ Appendix E.4a contains a figure which provides a complete overview of this discussion.

Therefore, the build block,s defining length is shortened/tightened: classical = 3 and new = 0. (See **Figure 22 Step 3.**)

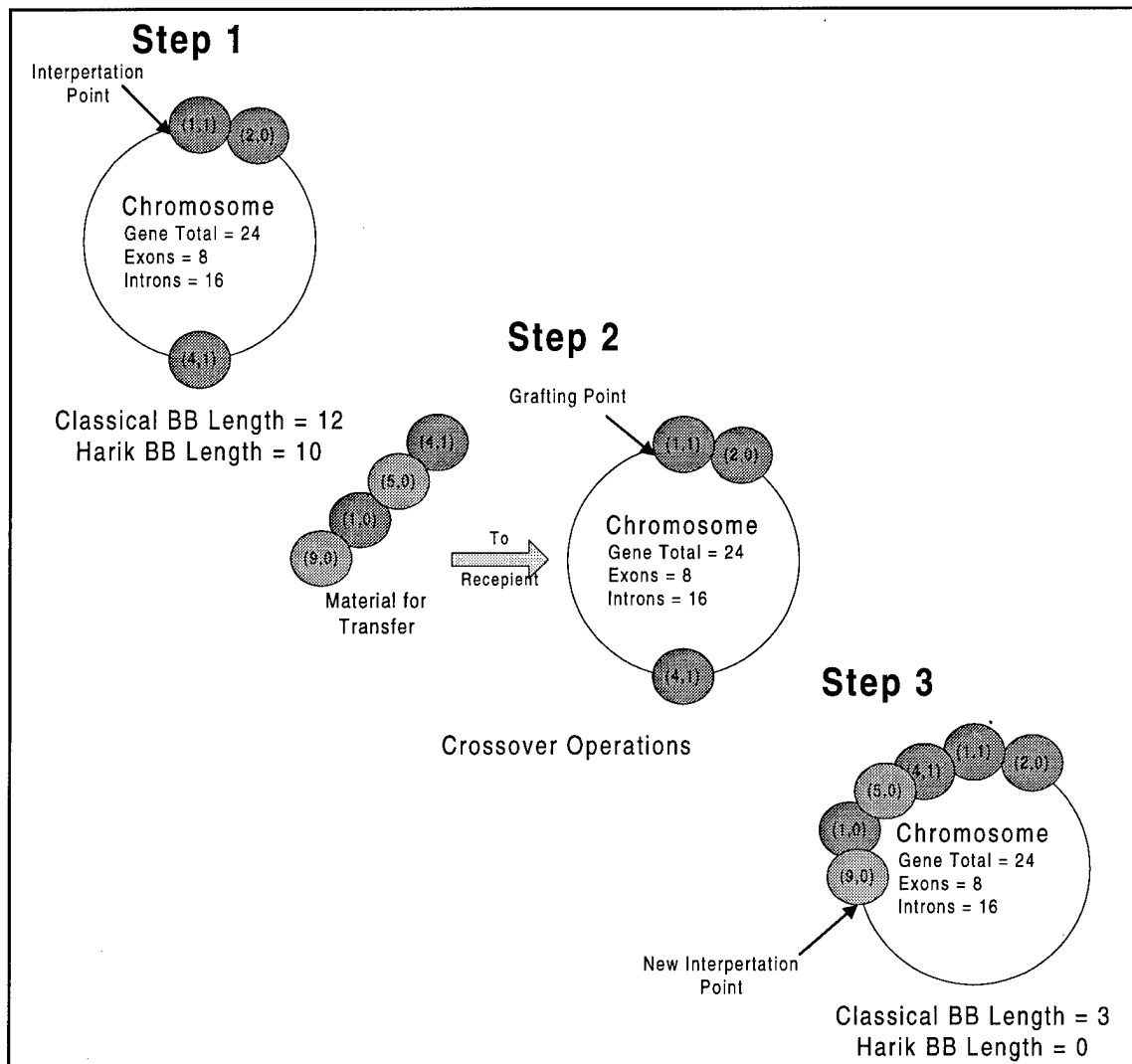


Figure 22: Crossover Operation Tightening of Building Blocks

Finally, the exchange operation is directional in that it has different effects on the donor and recipient chromosomes [7]. Harik suggests that this asymmetry can be remedied by having both individuals selected from the population play alternating roles and produce two offspring.

A high-level example of an LLGA algorithm is provided in the following figure:

```
Program LLGA

begin
  Initialization
  Fitness Evaluation
  Do
  {
    Tournament Selection      /* selection */
    Exchange Operation        /* crossover */
    Fitness Evaluation
  } Until (stopping condition is not reached)
end;
```

Algorithm 4: LLGA Pseudo-Code

3.7 Comparison

In this section, we compare the LIGAs based on initial population explosion problem, algorithmic complexity, and the order of linkage to which the algorithm could be successfully applied. This serves as the basis towards justifying why we did not implement three of the four LIGAs discussed. Furthermore, I point out some “pitfalls” of the algorithms that must be considered before they are applied to a real-world problem.

3.7.1 Initial Population Explosion Problem

We have defined the dramatic rate at which the initial population of any of these LIGAs increases as the “initial population explosion problem.” The initial population explosion problem affects the usefulness of the LIGA family in two aspects. The first aspect is its sheer size and the subsequent affect on the memory. For example, in order to investigate a 3rd-order linkage problem involving a 240 bit binary-chromosome ignoring chromosome representation overhead (i.e. the overhead of record structures, arrays, link list, etc.):

- The mGA requires 18,202,240 members in the initial population each, 240 bits long (see Equation 41). Since each gene is actually an allele/locus pair and the allele can be represented in 1 bit and the locus in 8, each gene requires 9 bits of storage space. Therefore, the population requires **39,316,838,400** bits (approximately 4.5 GBytes) of storage space during PEI.
- The SG algorithm only requires the number of allele values multiplied by the number of loci possible. For our example, the initial gene pool would have 480 genes (240 loci X 2 allele values). Each gene would be represented by 9 bits, assuming 1 bit for the allele and 8 bits for the locus, for a total of **4,320** bits.
- **Equation 18** implies that GEMGA requires the initial population to be 8 times the variation in fitness. If the variation of fitness amongst the initial population was 60,

then the initial population size would require 480 members. Each member has 240 genes requiring 9 bits for the allele/locus and 8 bits to represent the capacity for a total of **1,958,400** bits (1.96 Mbits). This figure does not include the number of bits required to represent the linkage set for each chromosome since it is dynamically growing.

- The LLGA needs an exponentially large number of introns coded into each chromosome as well as all allele complements for each locus to be present. Based on Equation 43 and Figure 76, we would need 4,560 introns and 480 exons in each chromosome. Since the LLGA makes no assumptions about the number of individuals required within the initial population, we have assumed 100 member based upon Harik's examples [7, 8, 17]. Including the introns, the space required is 9 bits per gene (same locus/allele representation) for 4,640 genes times 100 members which equals **4,176,000** bits (4.2 Mbits).

Even without the overhead of each particular LIGA's chromosome representation structure, the amount of memory required to contain the initial population is a severe requirement on the amount of available core memory. By accessing main memory to access the initial population, the processor's instruction execution rate is slowed down by disk access dramatically increasing the execution time of the algorithm!

The second aspect of the initial population explosion problem is the time spent conducting the initial fitness function evaluations. In this case, the fitness function takes 1.09119^{46} seconds to complete. Therefore:

- Since the mGA requires a fitness evaluation for each member of its initial population: 18,202,240 members X 1.09119 seconds = 19,862,102.27 seconds or **5,517.25 hours (229 days)**.
- Since the SG algorithm does not require an initial fitness for the gene pool because of the way it models the population, this aspect of the problem does not affect it.
- The GEMGA requires an initial fitness evaluation and creation of the linkages before it enters the main selection/reproduction phase of the algorithm: (480 members X 1.09119 seconds) + (480 members X 240 bits X 1.09119 seconds) = 126,229 seconds or **35.1 hours**.
- Since the LLGA requires a fitness evaluation for each member of its population: 100 members X 1.09119 seconds = 109.119 seconds or **0.3 hours**.

The advantage of not implicitly representing the population helps the SG GA to begin manipulating its "population" long before the other LIGAs have entered their main selection/reproduction phases.

One important consideration must be pointed out about the mGA. Whereas each of the other GAs requiring initial fitness evaluations use completely specified chromosomes, the mGA can have under/over-specified chromosomes. The over-

⁴⁶ 1.09119 represents the average time spent conducting a CHARMm evaluation using 35,100 trials.

specified chromosomes create no problems, but the under-specified ones need the competitive template in order for their fitness to be computed. The template has the effect of driving the under-specified chromosomes towards the phenotype represented by the template. If the global optimum is unknown and, therefore, not used in the construction of the template, then the template may drive the mGA population toward a suboptimal area of the search space. Therefore, we suggest that a random template be created each time an under-specified chromosome needs evaluation. By following this method, we would still reward the under-specified chromosome whose fitness is improved by the template and there won't be the tendency to drive the mGA's population toward any "predetermined" template/search space location. On the other hand, creating a "new" template for each under-specified chromosome incurs substantial overhead in large populations.

3.7.2 Algorithmic Complexity

The complexity of the GA is usually much less than the complexity of the fitness function in "real-world" applications. It is the fitness function's algorithm that typically drives the overall complexity of a GA once it is applied to a particular application. By comparing the complexity of the LIGA family of GAs, we can estimate a lower algorithmic complexity bound. If we combine this lower bound with the fitness function's algorithmic complexity, then we have a good range on the order of processing time we can expect from the GA. (i.e., The area between upper and lower bounds of a Big Θ complexity curve completely bound the expected runtime of the algorithm.)

Table 8 illustrates the complexity for each of the LIGAs covered in this paper.

Algorithm	Authors	Complexity	Key
Messy GA	Goldberg	$\Theta(l^k)$	Where l is the number of loci and k is the size of the building block [2]
Selfish Gene	Corno	$\Theta(l \times \text{cardinality}(\Lambda))$	Where l is number of loci and Λ is the alphabet [4].
Linkage Learning GA	Harik	$\Theta(l \times \sqrt{l} \times \ln(l))$	Where l is the length of the chromosome [17].
Gene Expression GA	Kargupta	$\Theta(l \times \text{cardinality}(\Lambda)^k)$	Where l is number of loci, Λ is the alphabet, and k is the size of the building block [10]

Table 8: Algorithmic Complexity

When plotted (see **Figure 23**), the mGA's complexity starts higher and grows at the worst rate. On the other end of the scale, the SG GA and GEMGA algorithms' complexity curves barely grow versus increasingly larger chromosome length. The LLGA is in the top half of the graph, and its growth rate is worse than either the SG GA or GEMGA.

It is important to note that the SG GA does not explicitly take into account the order of linkage we are trying to investigate. Instead, the SG GA iteratively evaluates the genes in the virtual gene pool uncovering higher order linkages in descending order of importance. This constraint hampers our ability to execute the SG GA for some fixed number of generations because there is no way to stipulate when the SG GA has uncovered the degree of linkage within the particular problem.

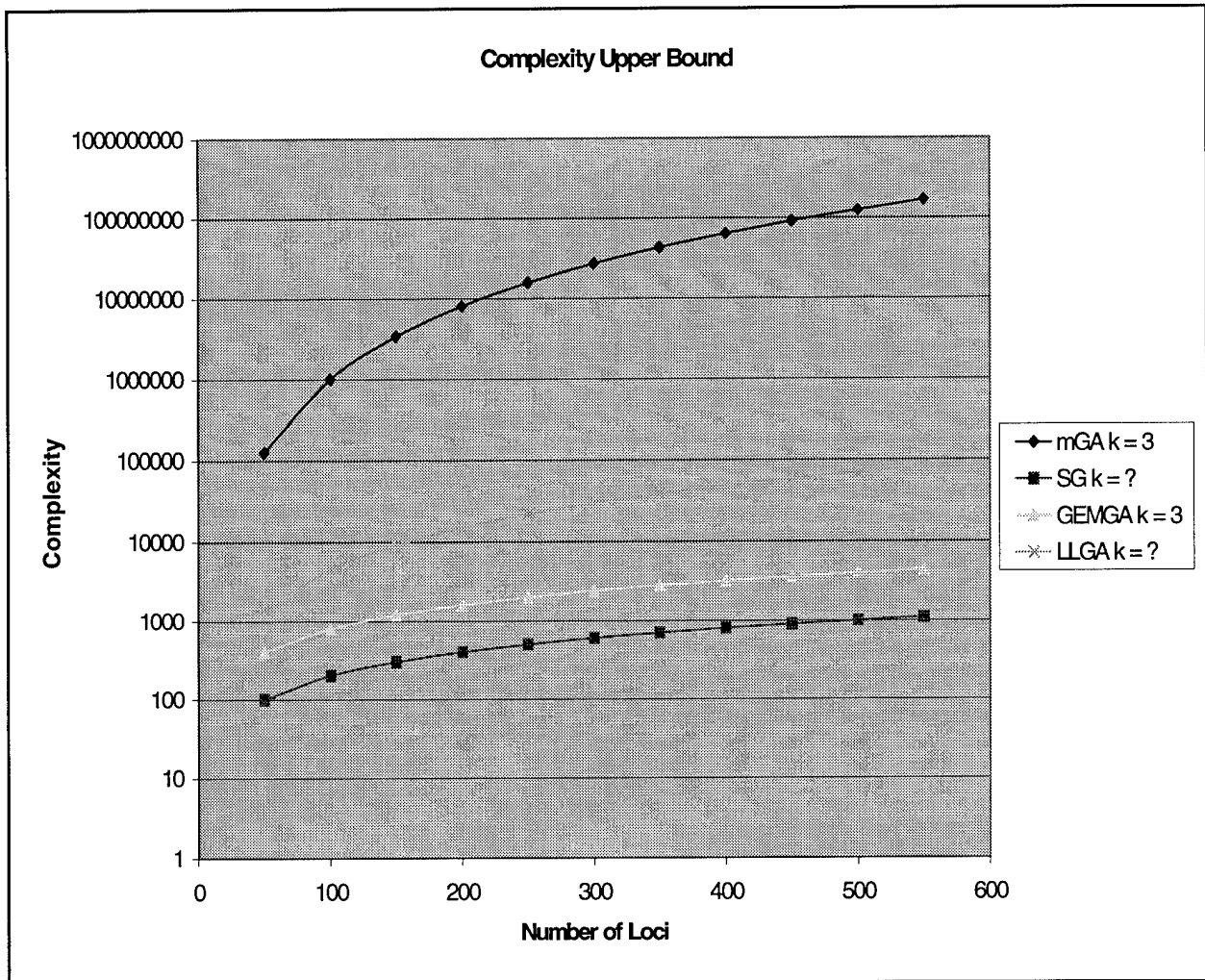


Figure 23: LIGA Complexity Curves⁴⁷

3.7.3 Linkage Order Ability

Finally, if we look at the order of linkage this family of GAs can investigate, we find that it is based upon the other two factors we have discussed: the initial population explosion and algorithmic complexity. For the mGA and GEMGA, the initial population fitness function evaluation process is the driving limitation. Neither of these two LIGAs are efficient for investigating 3rd-order linkages or higher because of the initial population fitness function evaluation execution time, 229 and ≈ 1.5 days respectively for the examples in the previous section.

The linkage uncovered by the SG GA is strongly dependent on the number of generations spent in replication and the size of epsilon chosen to reduce/increase the

⁴⁷ The y-axis is logarithmic, ℓ is the string length, k equals 3, and Λ is a binary alphabet.

allele frequency. The epsilon value drives the convergence of this algorithm, and Corno *et al* only suggestion for choosing a good epsilon⁴⁸ is experimentation [4, 5]. Therefore, we must continually execute the SG GA with smaller and smaller epsilons until the GA consistently finds the linkage we desire using a predetermined number of generations. If the number of generations changes, there is no guarantee the “good” epsilon value continues to perform as anticipated. Finally, although the time spent in initially evaluating the fitness of the initial population in the LLGA is a small deterrent for its use, the real hindrance is the vast number of introns requiring encoding within the chromosome. Harik has suggested compression methods for reducing the memory requirements of the LLGA chromosome, but for “interesting” problems this compression many not save enough memory to be fruitful (remember the example required 4,560 introns).

It is important to note that the success of GEMGA is based on two considerations that have so far not fit in this discussion. The first is the c-value. The c-value represents the variation of fitness amongst the individuals of the population, but our general impression is that the c-value is problem-domain-based, and therefore search-space-based. Since the initial population is randomly picked from the search space, the c-value is unknown prior to the complete initialization of the population, but the “c” value impacts the size of the population we need to combat a particular order-k deception (see Figure 74). This leads us into a “*which came first the chicken or the egg*” situation. We can increase the size of the initial population in order to decrease “c,” but when we do this, the variance amongst the population’s fitness decreases and requirement for increasing population size disappears. But, by decreasing the variance amongst the initial population, we are restricting the subspace GEMGA searches. Furthermore, when a population is comprised of nearly identical individuals, any GA quickly converges to the “optimal” value represented within the bound search area. This leads to preconvergence. The “c” value also impacts the amount of time the algorithm remains in the Transcription phase because there are more members of the population to evaluate. The other algorithmic consideration is that GEMGA only calculates the linkage sets once. Most of the interesting real-world problems do not conform to this static linkage concept, but they demonstrate dynamic linkage characteristics. (i.e., dynamic linkages change/mutate over the evolutionary process.) For instance, there

⁴⁸ epsilon is problem-domain-dependent

may be a 3rd-order linkage within a chromosome through out the evolutionary process of an organism, but as successive generations are evolved and die-off, this 3rd-order linkage may involve different genes.

3.7 Summary

All LIGAs are plagued by three problems: initial population explosion, algorithmic complexity, and low k-order of linkage investigation ability. How we incorporate problem domain knowledge into our LIGA greatly impacts the success of any search investigation. Any genetic algorithm that ignores the linkages between genes also ignores the evolutionary processes conceptualized in field of genetic algorithms. Only by modeling as many of the possible evolutionary processes as possible within our GA family of algorithms are we able to solve complex and interesting real-world problems. But, as this chapter has pointed out, the more we try to model, the more complex the process becomes. Currently, our best hope for solving interesting problems may lie with the LLGA or the SG GA with their small initial population and algorithmic complexity.

Table 9 summarizes the characteristics of each algorithm discussed.

Algorithm	Chromosome	Mutation	Crossover	Selection
mGA	Variable length	Random mutations allowed	One-point	Traditionally, tournament selection.
SG GA	None	None	None	Random construction of binary tournament between individuals drawn randomly from the virtual gene pool
GEMGA	Fully-specified and dynamically growing	None	Bit masking based upon the dominate donor	Random.
LLGA	Over-specified and includes introns	Implicit by re-orienting the interpretation point	Two-point	Traditionally, tournament selection.

Table 9: Algorithm Characterisitcs Summerized

Table 10 indicates some problem domains that these algorithms have been successfully applied towards (* indicates NP-complete problem). Chapter 4 discusses our integration of the LLGA with the Protein Structure Prediction (PSP) problem domain.

We decided not to integrate the SG GA with the PSP problem because of the inability to determine the order of linkage the algorithm will/did uncover and the required tuning of the epsilon value.

Problem Domain	MGA	SG GA	GEMGA	LLGA
0/1 Knapsack Problem*		X		
0/1 Multiple Knapsack Problem*		X		
The Sphere Model			X	
Griewank's Function			X	
Shekel's Foxholes			X	
Michalewicz's Function			X	
Langerman's Function			X	
Order-X Trap Function	X (order 3)		X (order 5)	X (order 4)
One-Max Problem			X	X
Muhlenbein Function			X	
Rosenbrock's Saddle	X			
Traveling Salesman Problem	X			
PSP/PFP	X			X

Table 10: Successful LIGA Applications

4.0 Genetic Algorithm (GA) Design and Implementation

4.1 Introduction

Previous research at AFIT resulted in a number of parallel and serial genetic algorithm implementations and evaluation functions for several domains [14, 15, 18, 37, 52, 66, 73]. Collectively, these are known as the AFIT Genetic Computation Toolkit (AGCT). The current state of the AGCT toolkit is in **Figure 24**. The contributions of this research to the toolkit are marked by (Deerman).

The purpose of this chapter is document the design decisions, implementation details, and interface requirements of the LLGA/CHARMm integration. A recurring situation throughout this research has been the restricted amount of prior design and implementation documentation details. Therefore, we have taken it upon ourselves to explain the in's and out's of the CHARMm energy model as implemented by AFIT, the redesign/integration of the LLGA to incorporate the PSP problem, and the integration of AFIT's CHARMm code with Ramachandran constraints as developed in Chapter 2. Our intent is to provide complete documentation relating to the design and implementation.

Section 4.2 documents the CHARMm energy model implementation developed by Brinkman [18] and refined by Gates [15]. **Section 4.3** rationalizes our redesign, implementation, and integration of the LLGA, and finally, **Section 4.4** provides our Ramachandran constrained CHARMm energy model. Where appropriate, design alternatives are indicated.

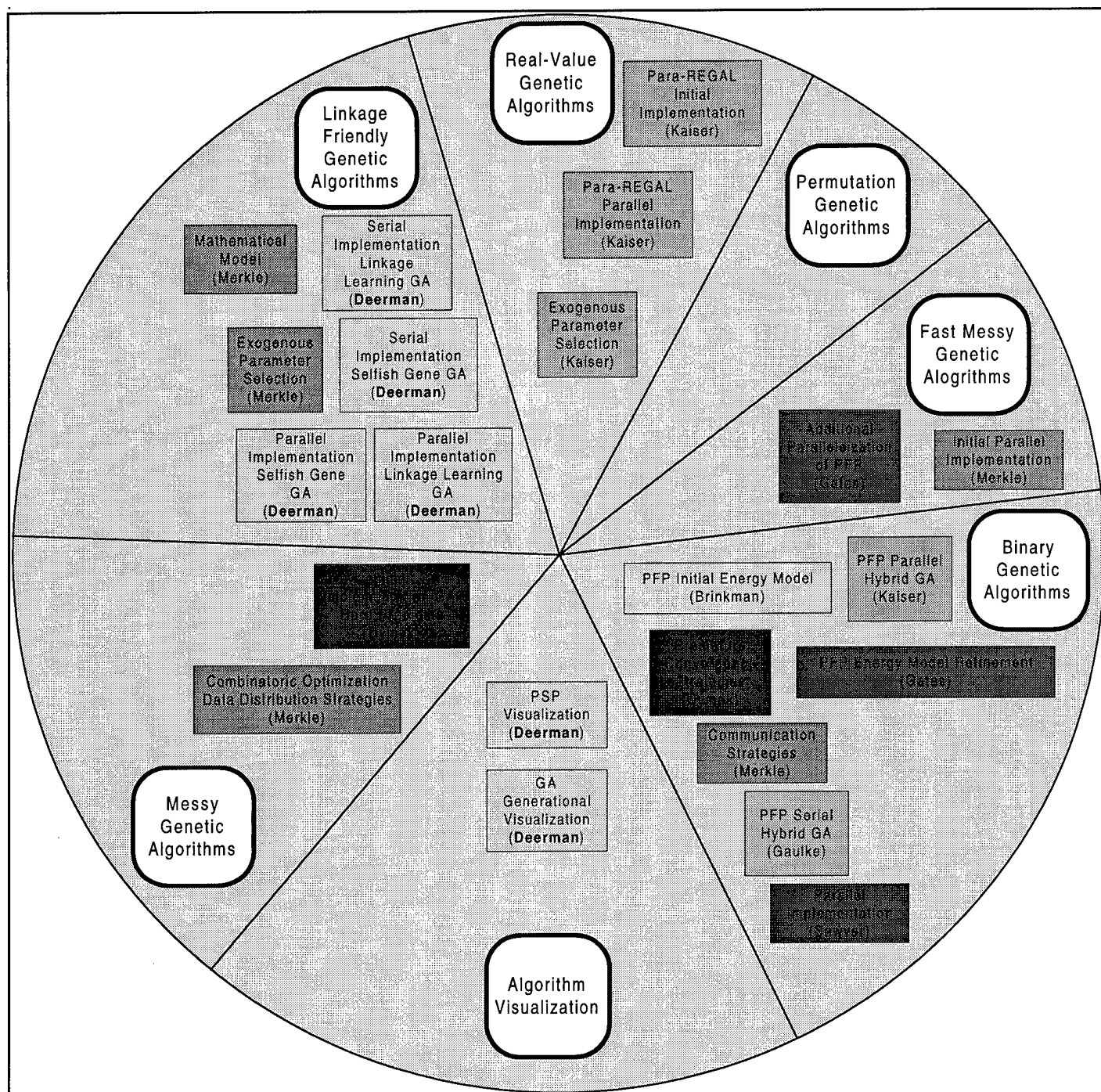


Figure 24: AGCT Genetic Algorithm Toolkit

4.2 CHARMM Implementation Design

The CHARMM code developed at AFIT follows the *Structured Analysis and Design* paradigm of software development [70]. This paradigm takes a top-down approach for partitioning the “problem” into subproblems which can be easily mapped to

specific implementable modules. Discussed separately in this section is the *how to's* of integration and the rough *control flow* for the implemented energy model.

4.2.1 Integrating CHARMM

In order to invoke the CHARMM energy model correctly, the GA initially calls *molecule(int length)* within *molecule.c*. The parameter "length" represents the number of characters that comprises the chromosome. From this module, the parameter file, the z-matrix file, and the RTF file are read and the CHARMM model is initialized. Once initialized, the GA calls *charmm_eval(string chromosome, int length)* which returns the calculated energy for the particular chromosome as a "C" double. *charmm_eval* is located in *energy.c*.

Another interface to AFIT's model is *domain_output(string chromosome, int length)* located in *charmm.sga.out.c*. This module calculates the energy just as *charmm_eval*; furthermore it produces an output file call *eval.bst* which contains a term-by-term breakdown of the energy calculation and the value of each independent dihedral angle. This module is employed for displaying the energy terms and angles for a single chromosome/protein.

4.2.2 Design Implementation

As presented in Error! Reference source not found., when a call to *charmm_eval* commences, the chromosome is initially decoded from its binary representation to its dihedral angles. Each dihedral angle is assumed to represent a particular radian value within the molecule, and each angle is specified by 10 binary digits allowing for the encoding of 1,024 different radian values per dihedral angle. For example, [Met]-Enkephalin is represented by the amino acids Tyrosine-Glycine-Glycine-Phenylalanine-Methionine, and it has 24 dihedral angles. Between each amino acid there are three or more dihedral angles depending on whether a side chain corresponds to the particular amino acid⁴⁹.

⁴⁹ The tyrosine has 3 side chains, phenylalanine has 2 side chains, and methionine has 4 side chains. See **FIGURE**

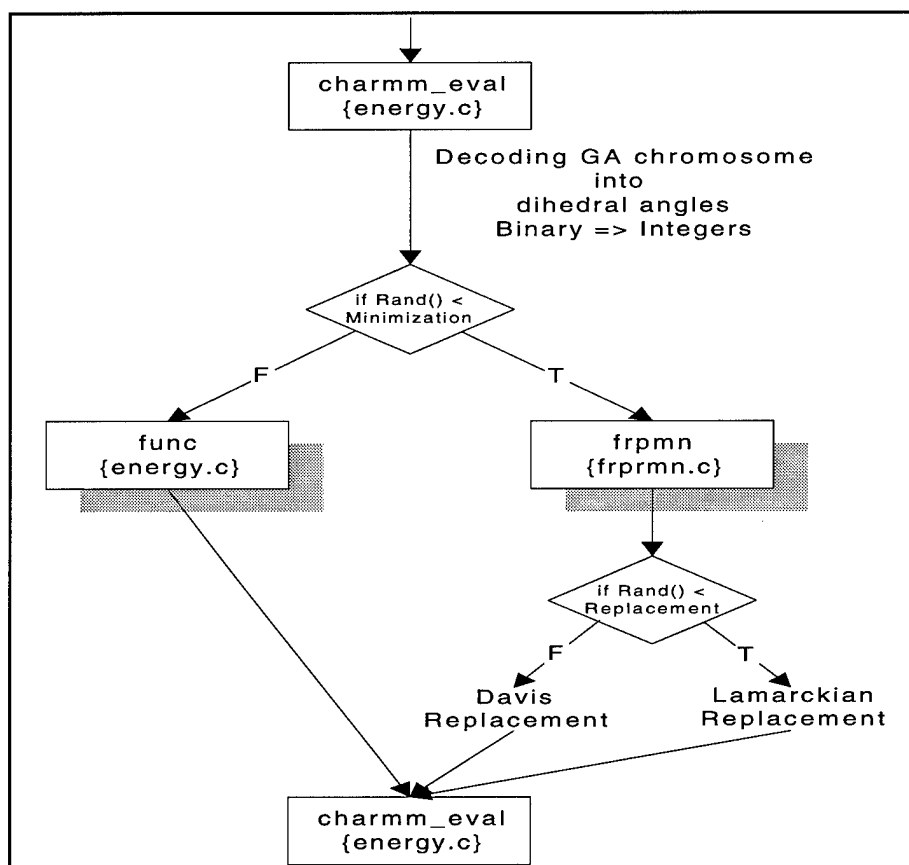


Figure 25: CHARMm Source Code Control Flow

Thus, the GA chromosome is 240 binary values (i.e., 24 ten bit binary numbers). The GA chromosome represents these dihedral angles in accordance with the z-matrix input file. The z-matrix file is a sequential listing of all atoms present in the molecule. See **Figure 26**.

atom	bond length	flag	bond angle	flag	dihedral	usage flag	atom _j	atom _k	atom _l	charge
------	-------------	------	------------	------	----------	------------	-------------------	-------------------	-------------------	--------

Figure 26: Z-matrix Format

The **atom** field represents the atom in the protein from which the **bond length**, **bond angle**, and **dihedral** fields are calculated. Therefore, the bond length is the distance between atom and atom_j. The bond angle is a radian measurement of the angle formed by the atom, atom_j, and atom_k. The dihedral is the torsion angle in radians of the middle bond formed by the atom, atom_j, atom_k, and atom_l. But the key to the z-matrix file is the **usage flag**. This flag specifies whether dihedral angle is dependent and independent. If the usage flag is set to 1, the angle formed by atom, atom_j, atom_k, and atom_l represent an independent dihedral angle that is the principle dihedral angle

used in the AFIT CHARMM energy calculations. If the usage flag is set to 2, indicating a dependent dihedral angle, or 0, indicating that this dihedral is not used in the energy calculation, then it is not represented within the GA chromosome.

Figure 32 is a representation of [Met]-Enkephalin corresponding to the z-matrix filed named nayeem.z. This z-matrix file is in the “correct” configuration for the AFIT CHARMM energy model. The atoms as numbered in **Figure 32** correspond to the “atom’s”, $atom_j$, $atom_k$, and $atom_l$ in nayeem.z. If this z-matrix file is used with the corresponding parameter file (PARM.PRM) and topology file (NAYEEM.RTF) located in ~genetic/inputfiles, then the associated input energy calculated by AFIT’s model matches **Table 11**. The dihedral angles for the [Met]-Enkephalin molecule depicted in **Figure 32** that energies are given by **Table 11** are in **Table 14**.

TERM	ENERGY
Fixed bond energy	12.380356
Dependent bond energy	0.000000
Independent bond energy	0.000000
BOND ENERGY	12.380356
Fixed angle energy	6.189469
Dependent angle energy	0.000000
Independent angle energy	0.000000
ANGLE ENERGY	6.189469
Fixed dihedral energy	0.000160
Dependent dihedral energy	5.415865
Independent dihedral energy	2.787972
DIHEDRAL ENERGY	8.203997
Lennard-Jones energy	-18.802334
1-4 L-J interaction energy	3.163984
LENNARD-JONES ENERGY	-15.6383349
Electrostatic energy	-88.855370
1-4 electrostatic energy	48.752795
ELECTROSTATIC ENERGY	-40.102574
NON-BONDED ENERGY	-55.740923
TOTAL ENERGY	-28.967101

Table 11: Correct Energy Values Associated with the Correct Z-Matrix File

The translation between the z-matrix file layout to the GA chromosome is esoteric and solely dependent upon the ordering of the principle/independent dihedral angles as defined by the z-matrix file. **Table 12** depicts z-matrix to GA chromosome corresponds to the nayeem.z file layout.

Amino Acid	Tyr	Tyr	Gly ₁	Gly ₁	Gly ₂	Gly ₂	Gly ₂	Phe	Phe	Met	Tyr	Phe	Met	Met	Met	Tyr	Met	Met						
Dihedral Angle	Φ	Ψ	Φ	Ψ	Φ	Ψ	Φ	Ψ	Φ	Φ	χ ₁	χ ₂	χ ₁	χ ₂	χ ₃	Ψ	χ ₃	χ ₄						
Starting Bit in the Chromosome	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230

Next, the decoded chromosome is either locally optimized or its energy value is just calculated. The constant, *Minimization*, determines which path to traverse. The lower its value the less likely the evaluation performs local optimization. Local optimization takes the form of either a Baldwinian (Davis replacement⁵⁰) or Lamarckian approach. In the Lamarckian method, the chromosome and its fitness value is replaced by the best locally optimized chromosome. The Baldwinian approach, on the other hand, just replaces the fitness value of the passed chromosome with the fitness of the best locally optimized chromosome. Each of these techniques requires at least three executions of the CHARMM energy model.

Appendix G contains data flow diagrams for the complete CHARMM energy model. These diagrams document the design of the model as developed by Brinkman [18] and Gates [15].

4.3 LLGA/PSP Design and Implementation Details

The LLGA is developed following an object-oriented methodology. The object-oriented approach to software engineering is based upon the modeling of objects from the “real world” and then using the model to build a language-dependent design organized around those objects [71]. Object-oriented practitioners argue that the paradigm promotes better understanding of requirements, cleaner design, and a more maintainable system [71]. Consequently, it was our goal not to re-engineer the given class interfaces or overall object design.

A few additions to the overall design were required to integrate Harik’s implementation with our problem domain. **Figure 27** shows the complete LLGA class hierarchy⁵¹. The classes “Bbtemplate,” “Worst,” and “timing” were added. In total, five major challenges needed to be met in order to integrate the LLGA with the PSP problem domain. Each challenge is discussed separately.

⁵⁰ In the code, it is called Davis Replacement not Baldwinian. Therefore, we choose consistency with the code over exactness.

⁵¹ **Appendix H** contains the Rumbaugh diagrams for each object.

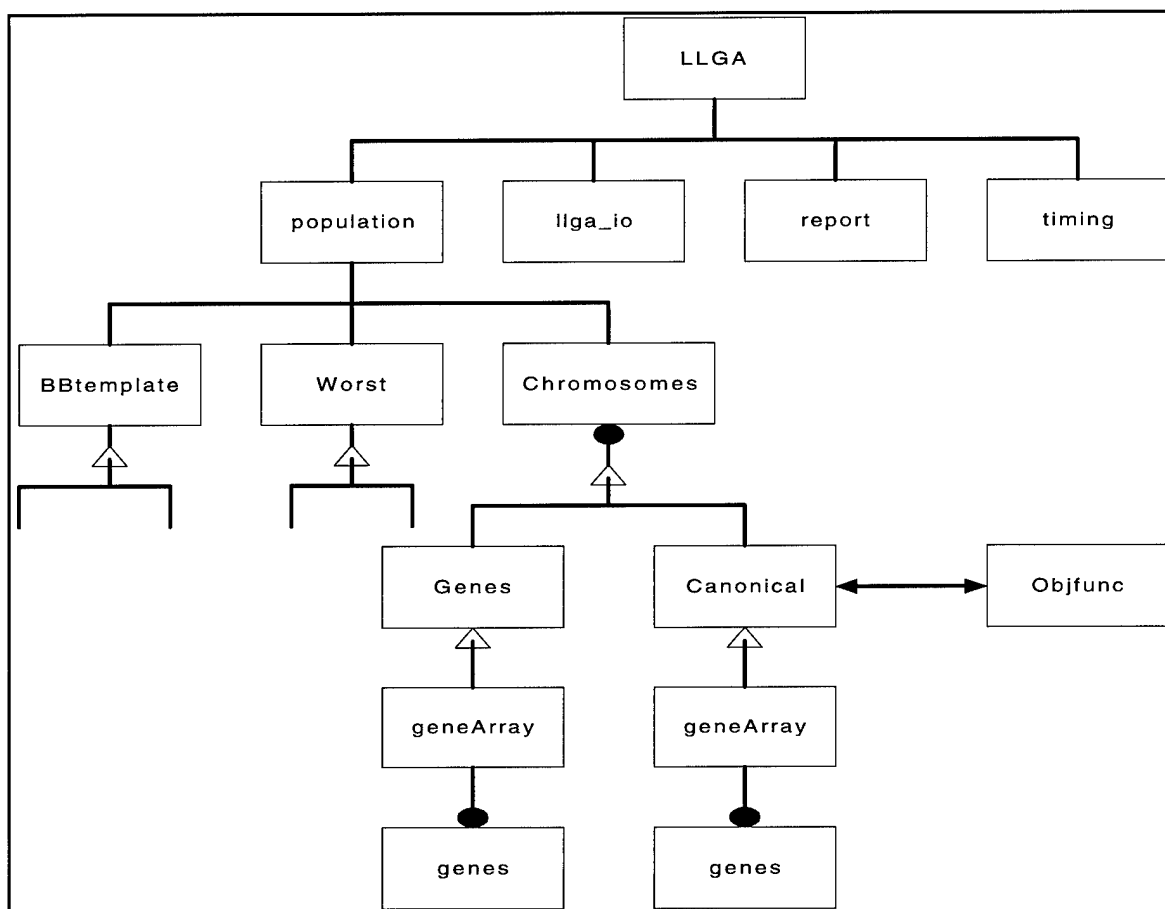


Figure 27: LLGA Class Hierarchy

4.3.1 Challenge 1: Building Block Assumption

In Harik's original LLGA implementation, he assumed that the perfect building block (BB) contains all "1's" in each allele. This is an unreasonable assumption for any real problem. In particular, a [Met]-Enkephalin chromosome constructed from just 1's represents a fitness of 12.980 kilocalories which is far above the the QUANTA™ minimum of -29.225 kilocalories [15]. We conjecture that Harik made this assumption because he used the Max-Ones problem in order to show the power of the LLGA in his dissertation [17]. The Max-Ones problem is a deceptive problem in which the fitness a certain local maximum (represented by all 0's) approaches that of the global maximum (represented by all 1's) [17]. Therefore, by "hard coding" his algorithm, he could drive his solution to the global maximum!

To correct this situation and to generalize the algorithm, we included a new attribute of a population called a BBtemplate (i.e. building block template). The BBtemplate is loaded from a file during the creation of the population. On the other

hand, the BB template is never used to control the direction of search in either the original LLGA or our modified version. The BB template's only function is as a comparison tool used during reporting! Each chromosome in the population is compared to the BB template to determine the number of BBs contained within the population. If the global minimum/maximum is known, then the BB template can serve as a compass indicating how close the GA is to the global minimum/maximum. Otherwise, any string of 1's and 0's can represent the BB.

Two other alternatives were evaluated before implementing the alternative discussed above. The first alternative used a randomly generated chromosome as the BB under the assumption that since for most real world problems we do not know the optimal solution, a random guess at the BB distribution was as good as any. This is the simplest solution, and probably the less likely to produce a "good" solution because hopefully there is some educated guess from the problem domain which we could incorporate into our approach. The next alternative we investigated involves a dynamic BB that would start as a randomly generated BB. Then, as better solutions are encountered, the BB template is updated to reflect the changing landscape. We conjectured that this solution would be optimal in the situation that the BB within the chromosome changed without increasing in number. For instance, it is conjectured that the PSP problem relies upon 5 BB [15]. We also anticipate that as the protein folds too more and more compact states, the dihedral angles included within the BB changes. On the other hand, if we are constantly changing the BB template to match the optimal individual, we may be driving the LLGA to a local minimum. Consequently, we chose not to implement this method.

4.3.2 Challenge 2: Recording the Optimal Solution Uncovered

The original LLGA contained no process by which to record and report the optimal individual found. It is possible for a GA to find the global optimum answer during initialization, and then breed this chromosome out of the population during future generations due to the destructive effects of crossover and mutation. Therefore, we added a new attribute to the population called "worst⁵²" that records the worst chromosome found across generations. When our LLGA is initialized, we randomly choose the worst chromosome from the problem space. Then, at the end of each generation, we compare the worst chromosome to each member of the population. If a

population member has a fitness value worse then the previous worst chromosome, it replaces the previous worst in the next generation. Once the GA has terminated, the worst ever chromosome is reported with its fitness value.

4.3.3 Challenge 3: Integration of CHARMM

The integration of the CHARMM energy model into the LLGA was nearly trivial. Harik's class decomposition allows for the inclusion of additional fitness functions in the *Objunc* class. Basically, the *objfunc()* needed to be changed to point to a new function called *CHARMM_EVAL()*. Furthermore, the reporting functions in *llga_io.cpp* needed to be updated to include the new fitness function name. Since we made the decision not to completely re-engineer Harik's interface methodology, *CHARMM_EVAL()* parameter list contains three dummy variables which was necessary for the *llga_io* reporting function's look-up table to recognize the new signature⁵³.

4.3.4 Challenge 4: Parallel Implementation and Execution Timing

It is one of the goals of this research to parallelize the LLGA and report on the efficiency of the implementation. For parallelization, we used the message-passing interface (MPI) standard to implement a master-slave-farming model⁵⁴. It is the master's responsibility to conduct the "normal" GA operations of reproduction and selection. The master also controls the distribution of the fitness evaluations to each of the slaves. In this model, the master usually remains idle while the slaves do their work. This leads to an unbalanced distribution of the work per unit time which is undesirable because it leads to under utilization of the processors. To curb this situation, the master also has the responsibility of evaluating the "worst" chromosome during the initial generation. There after, the fitness of the worst chromosome is copied during the recording of the "new" worst.

The partitioning of the data is accomplished using three messages per generation per processor. The first message from the master to the slave indicates that slave is about to be put to work. The second message transmits the chromosomes the slave needs to evaluate and the last message returns the evaluated chromosomes back to the master. The major challenge in implementing this scheme was the construction

⁵² We choose to call the attribute "worst" because intuitively the minimum value is the worst.

⁵³ All fitness function signatures needed to be consistent.

of the message. MPI does not intrinsically handle the transmission of C++ objects nor does it handle composite data structures. In order to transmit a C++ object, we built an MPI derived message type based upon the decoded chromosome object's atomic C++ data structures [72, 84]. The end result is less communication overhead, which decreases the parallel LLGA's (pLLGA) execution time. The decrease of communication results from not transmitting the chromosome string and its associated fitness field as separate messages.

A final note concerning the parallelization of the LLGA must be discussed. The pLLGA requires an odd number of processors, and the number of slaves must evenly divide the members of the population without a remainder. We placed this last restriction upon the pLLGA in order to simplify the algorithm's implementation. Without this restriction the master processor would have to unevenly distribute the workload. This in turn would cause load-in-balance for the homogenous parallel platform, and the need to investigate and implement an appropriate load-balancing scheme for the heterogeneous AFIT Beowulf.

The efficiency of the LLGA has never been reported in terms of minutes/hours. We chose to implement a separate class devoted to the capture and reporting of execution times for the LLGA. Two aspects of the LLGA are report. The first is the total time the LLGA is executed. Secondly, the total time spent accomplishing fitness evaluations, the average per fitness evaluation, and the number of fitness evaluations is also reported. This information indicates how much time the pLLGA spent conducting parallel operations. An alternative approach would be to use global variable to capture the timing, but global variable are **NOT** an appropriate programming construct under any circumstances.

4.3.4 Challenge 5: Random Number Generator Correctness

The stochastic nature of a GA is totally dependent upon the implemented random number generator. Dymek's Appendix A [66] covers the importance of random number generators due to this heavy reliance. The random number seed dictates where in the problem's search space the GA begins searching. Therefore, it is extremely important that "good" random number generators are used. A good random number generator is defined as one in which no "perfect correlation" occurs [66]; a

⁵⁴ Models of parallelization are discussed in **APPENDIX D**. The LLGA follows the data decomposition model.

perfect correlation between two random number generators results in the same instantiated behavior from two separate GA executions. At first glance, this sounds much worse than the situation warrants. For the purposes of validation of experiments, two separate GA test runs, which start with the same random number seed, should result in precisely the same GA behavior.

On the other hand, random number generators are only pseudo-random. The randomness of the sequence of generated “random” numbers depends on the seed. It is possible that two **distinct** seeds used to initialize a random number generator can result in the same or over lapping sequence of random numbers. This results in two separate GA executions having similar behavior, even though different random seeds are used. This is not desired here. Thus, the random number generator needs to be checked to ensure: 1) that the random numbers produced represent a uniformly distributed set of numbers between the lower and upper bound, 2) that the different seeds used in testing do not correlate, and 3) that within a series of random numbers there is no correlation indicating a relationship between the current random number and a previously generated random number. Uniform distribution guarantees that the random numbers generated have an equal chance of occurrence, as those that are not generated. If two separate random seeds produce correlating sequences of random numbers, two separate executions of the GA using these seeds would search the same problem domain landscape. Finally, we do not want a correlation within a sequence of random numbers because then our random sequence becomes predictable.

Figure 28 shows the results of evaluating the distribution of Harik’s random number generator using the first seed in **Table 17**.

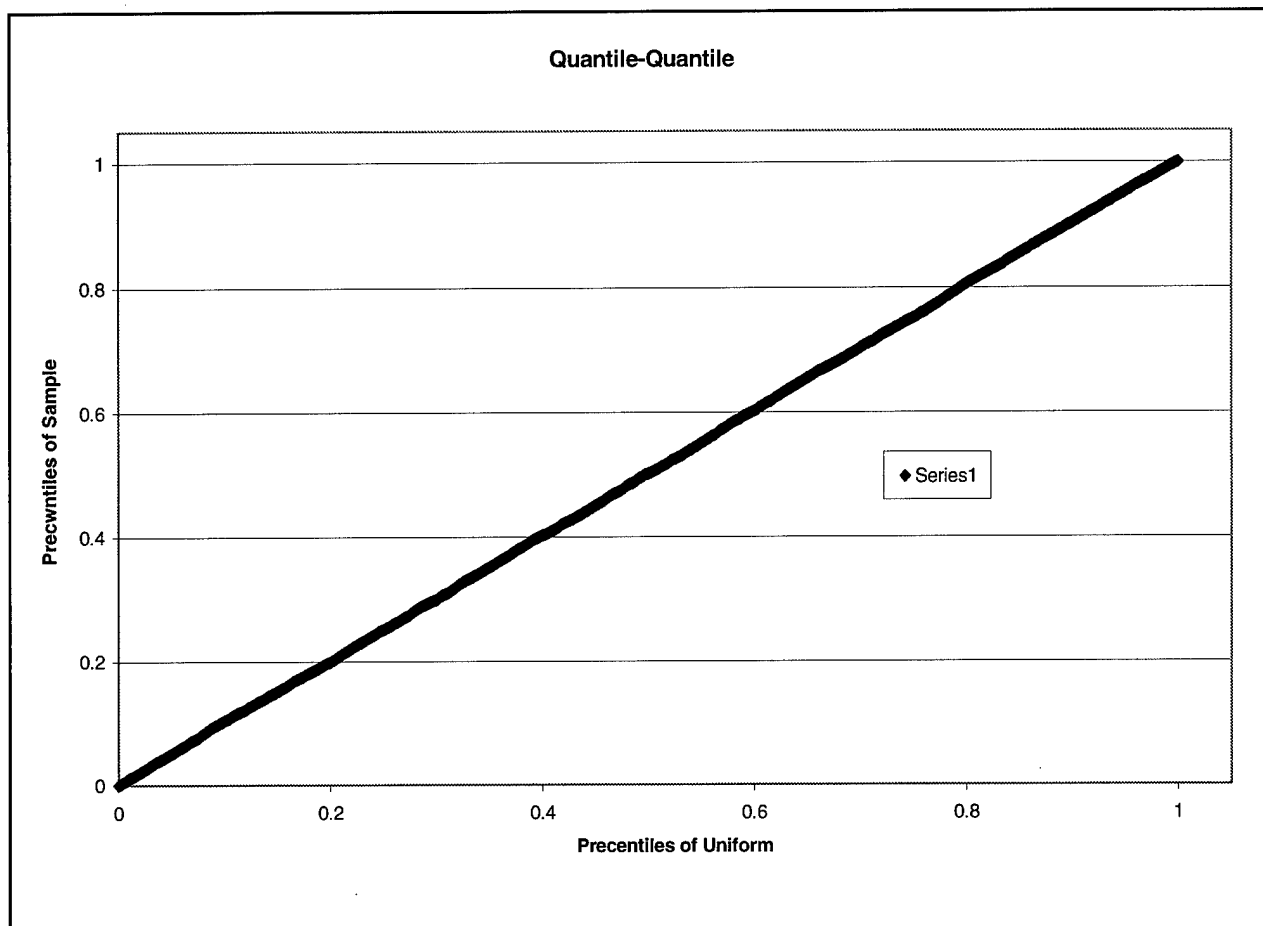


Figure 28: Uniformly Distributed Random Numbers

Figure 28 represents a uniform distribution because if it were not the graph would have a stepping characteristic which would indicate that a particular random number was generated two or more times. **Figure 28** was generated using the random numbers manufactured during the creation of the chromosomes and the determination whether or not to perform crossover.

Secondly, in order to determine if any two series of random sequences using unique seeds result in a correlation as series of pairwise correlations were calculated and tabulated along with their associated p -value. **Table 13** indicates the results. Out of the twenty different possibilities, there are only two cells that may indicate a possible correlation. There *may be* a correlation between the sequences of random numbers produced by seeds (3 and 5) and (2 and 7). On the other hand, because we are using a random number generator, it is possible that two sequences approximate each other *slightly* as in our case (only $1/10^{\text{th}}$ of the total number of comparison possible correlate). If **Table 13** indicated that there were correlations between more of the sequences (for

instance 25% of the cells indicated possible correlations), then we would be concerned that the random number generator was not producing random sequences of numbers.

Sequence	1	2	3	4	5	6	7	8
1	1	—	—	—	—	—	—	—
2	-0.00370 (0.71132)	1	—	—	—	—	—	—
3	0.00670 (0.50281)	-0.00593 (0.55314)	1	—	—	—	—	—
4	0.00599 (0.54934)	0.00544 (0.58679)	-0.00342 (0.73217)	1	—	—	—	—
5	0.00428 (0.66843)	0.00493 (0.62227)	0.02151 (0.03151)	0.01951 (0.05103)	1	—	—	—
6	-0.00019 (0.98526)	-0.00949 (0.34252)	0.01094 (0.27394)	-0.00873 (0.38284)	-0.00137 (0.89086)	1	—	—
7	0.00504 (0.61402)	-0.02191 (0.02844)	-0.00107 (0.91431)	-0.00150 (0.88097)	0.00695 (0.48706)	-0.01192 (0.23329)	1	—
8	-0.00410 (0.68168)	-0.01333 (0.18250)	0.01704 (0.08841)	-0.00610 (0.54175)	0.00229 (0.81914)	0.00905 (0.36543)	-0.00016 (0.98672)	1

Table 13: Pairwise Correlation

Finally, the sequence produced by each unique seed was checked to ensure that there was no dependence between any generated random number. The following graph is one example autocorrelation graph representative of each of the eight sequences to a lag of 500 [85]. The “lag” indicates sequential relationship between the random numbers compared for the correlation⁵⁵.

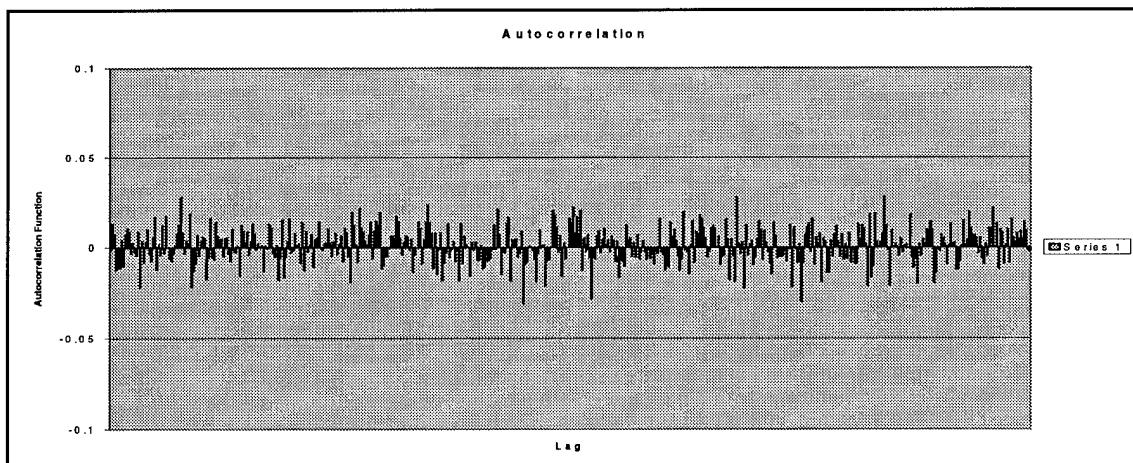


Figure 29: Autocorrelation for Random Seed 1

Figure 29 indicates that there is no correlation between the numbers generated. If a correlation were found, it would be represented as a repeating pattern within the

⁵⁵ A lag 500 test for a correlation between random numbers generated from the x^{th} number up to its 500th neighbor.

graph. The pattern could be on alternating sides of the x-axis (i.e. like to a sine/cosine/tangent wave) or repetitive on one side of the x-axis either above or below. None of the random seeds' autocorrelation indicated a correlation.

Therefore, together these three tests illustrate that the pseudo-random number generator developed by Harik behaves in a random fashion.

4.4 AFIT CHARMM Inclusion of Ramachandran Constraints

As stated in **Section 4.2**, when a chromosome is passed to *charmm_eval()*, it is first decoded. This decoding process is highly dependent on the current encoding of the chromosome in the GA. This encoding is implicitly defined in **Table 12**, and it is different for different molecules. Therefore, we choose to implement our constraints methodology as a conditional compilation scheme by employing the C *#ifdef* construct. The original code is shown in **Figure 30** and our modified code is in **Figure 31**.

```

indexPtr = Indep_dihedral;
while (indexPtr != NULL)
{
    temp = (double) Ctoi (&buff[start],
slice);
    P[n] = ((temp / max_range) * twoPI) -
PI;
    n++;
    start = start + slice;
    indexPtr = indexPtr->next;
}

```

Figure 30: Original Chromosome Decoding

In the previous implementation, each dihedral angle is translated from its 10-bit binary encoding to a radian value (a C double) between 0 and 2π . In the new constrained CHARMM decoding, the dihedral is decoded and then mapped to the appropriate constrained subrange of possible values depending on which dihedral it represents.

```

indexPtr = Indep_dihedral;
while (indexPtr != NULL)
{
    temp = (double) Ctoi (&buff[start], slice);
    temp = ((temp / max_range) * twoPI) - PI;
    #ifdef Met-Enkaphalin
    switch (n)
    /* non-glycine phi */
    case 1: case 10: case 13:
        temp = temp*((-210 - -30)/twoPI) + -210;
        break;
    /* glycine phi */
    case 4: case 7:
        temp = temp*((-315 - -45)/twoPI) + -315;
        break;
    /* psi */
    case 2: case 5: case 8: case 11: case 21:
        temp = temp*((210 - -90)/twoPI) + 210;
        break;
    /* omega */
    case 3: case 6: case 9: case 12: case 24:
        temp = temp*((-160 - -200)/twoPI) + -160;
        break;
    /* chi1&3 tyrosine, chi2&3 methionine */
    case 14: case 22: case 19: case 20:
        temp = temp*((-160 - -200)/twoPI) + -160;
        break;
    /* chi2 tyrosine, chi1 pheny, chi1 methionine */
    case 15: case 16: case 18:
        temp = temp*((75 - 45)/twoPI) + 75;
        break;
    /* chi2 pheny, chi4 methionine */
    case 17: case 23:
        temp = temp*((-75 - -45)/twoPI) + -75;
        break;
    default: printf("error in n = %i",n); exit(1);
    #endif
    P[n] = temp;
    n++;
    start = start + slice;
    indexPtr = indexPtr->next;
}

```

Figure 31: Modified Chromosome Decoding

The *switch* statement implements the transformations discussed in Chapter 2. This transformation process needs to be encoded for each different protein because the

chromosomal encoding is different. Newman projections for each constraint helps in visualizing the allowable regions for each dihedral angle, reference **APPENDIX I**.

4.5 Summary

Software reuse and portability have continued to be the driving force behind AFIT's development of code. The design presented in this chapter has maintained these objectives by integrating the object-oriented LLGA code with AFIT's own functionally decomposed CHARMM energy model without major modifications to either system. Furthermore, our novel incorporation of constraints can be easily disengaged if the inclusion of constraints into the energy model prove to be fruitless. The next chapter presents the engineering tests used to evaluate our modifications to the CHARMM energy model and Harik's algorithmic approach.

5.0 Design of Experiments

In the process of studying the protein structure prediction (PSP) problem, we have read many papers touting their experimental prowess, but if the truth were told, very few computational researchers conduct objective *experiments* with the basic *scientific method* in mind. A well-developed scientific experiment encompasses the following characteristics: a measurable objective/goal (hypothesis), well-defined methodology/procedures, validated results, and a logical conclusion(s). Conclusions may support or contradict the objective, but either outcome leads to useful information being provided. The scientific method consists of four repeated steps: observe, hypothesize, predict, and test. For instance, if we want to prove “A” is true, we can assume “A^o” is true. Observe the nature of A^o, and look for evidence that it is actually not true. If “strong” evidence exists, we can conclude that A^o is false and A is true. On the other hand, if weak or no evidence exist, then we must continue to assume A^o is true. But this does not prove A^o is true. We can always restate our hypothesis and re-evaluate “A” until what we want to prove is clearly valid or invalid. The scientific method combined with objective scientific experimentation results in a sound irrefutable⁵⁶ conclusion. For an in depth look at developing scientific experiments the reader is referred to [75].

For the purpose of this research, the observed phenomenon is the *protein structure prediction problem*, and the hypothesis is that *the parallel Linkage Learning Genetic Algorithm (LLGA) family⁵⁷ can generate an “acceptable⁵⁸” molecule conformation, by employing the CHARMM energy model and local minimization techniques as the fitness functions, more efficiently than previously employed AFIT methods.*

This chapter discusses how to test the LLGA (with and without domain constraints), described in **Chapter 3** and developed in **Chapter 4**. **Section 5.1** covers the proteins used. **Section 5.2** describes the general data requirements and statistical tests that are conducted. Finally, **Section 5.3** establishes each of the experiments performed to test the hypothesis.

⁵⁶ Irrefutability implies that your conclusions cannot be proven false.

⁵⁷ The LLGA family = LLGA, pLLGA, constrained-LLGA, constrained-pLLGA.

⁵⁸ Acceptability is defined here as a GA calculated protein with an RMSD of less than 1.

5.1 Test Molecules

Two separate protein molecules are used for these tests. The first molecule, [Met]-Enkephalin, is a very small polypeptide with only five amino-acid groups: Tyr-Gly-Gly-Phe-Met⁵⁹ using neutral NH₂ and -COOH as terminators at the α -amino and α -carboxyl ends, respectively. This protein was chosen because it has a confirmed conformation using the QUANTATM package and was used by relevant research efforts [6, 7, 8, 9, 14, 15, 18, 37, 52, 66, 73]. The second molecule, Polyalanine₅₇, was chosen because of its affinity to nicely fold into a α -helical structure. Polyalanine₅₇, a larger polypeptide than [Met]-Enkephalin, is defined by 57 amino-acid groups: Ala-Ala-Ala-...-Ala⁶⁰. We have chosen to use the same end groups as the [Met]-Enkephalin molecule. **Figure 32** and **Figure 33** are representation of [Met]-Enkephalin and Polyalanine₅₇, respectively. The figures are labeled to distinguish the dihedral angles along their molecular backbone. **Table 14** and **Table 15** outline the "correct" dihedral angles values for the "accepted"⁶¹ energy minimum defined by QUANTATM. The conformation energy for [Met]-Enkephalin is -29.225. Alternative molecules have been considered, (e.g., Crambin [79], P27-4 [80], P27-6 [80], P27-7 [80], cellular acid binding protein I [81], cucumber stellacyanin [81], endoglucanase [81], histidine-containing phosphocarrier protein [81], ubiquitin conjugating enzyme [81], and the Abl-SH3 domain of tyrosine kinase protein [62]). These molecules were not used because of their large size and because they have no accepted minimum conformation at this time.

Residue	Dihedral Angle (degrees)						
	Φ	Ψ	ω	χ_1	χ_2	χ_3	χ_4
Tyr	-86	156	-177	-173	79	166	—
Gly	-154	83	169	—	—	—	—
Gly	84	-74	170	—	—	—	—
Phe	-137	19	-174	59	-85	—	—
Met	-164	160	-180	53	175	-180	-59

Table 14: Dihedral Angles for [Met]-Enkephalin Accepted Energy Minimum

⁵⁹ Tyrosine-Glycine-Glycine-Phenylalanine-Methionine

⁶⁰ Alanine times 57.

⁶¹ Different molecular energy calculation engines may compute different energy values for the same molecule.

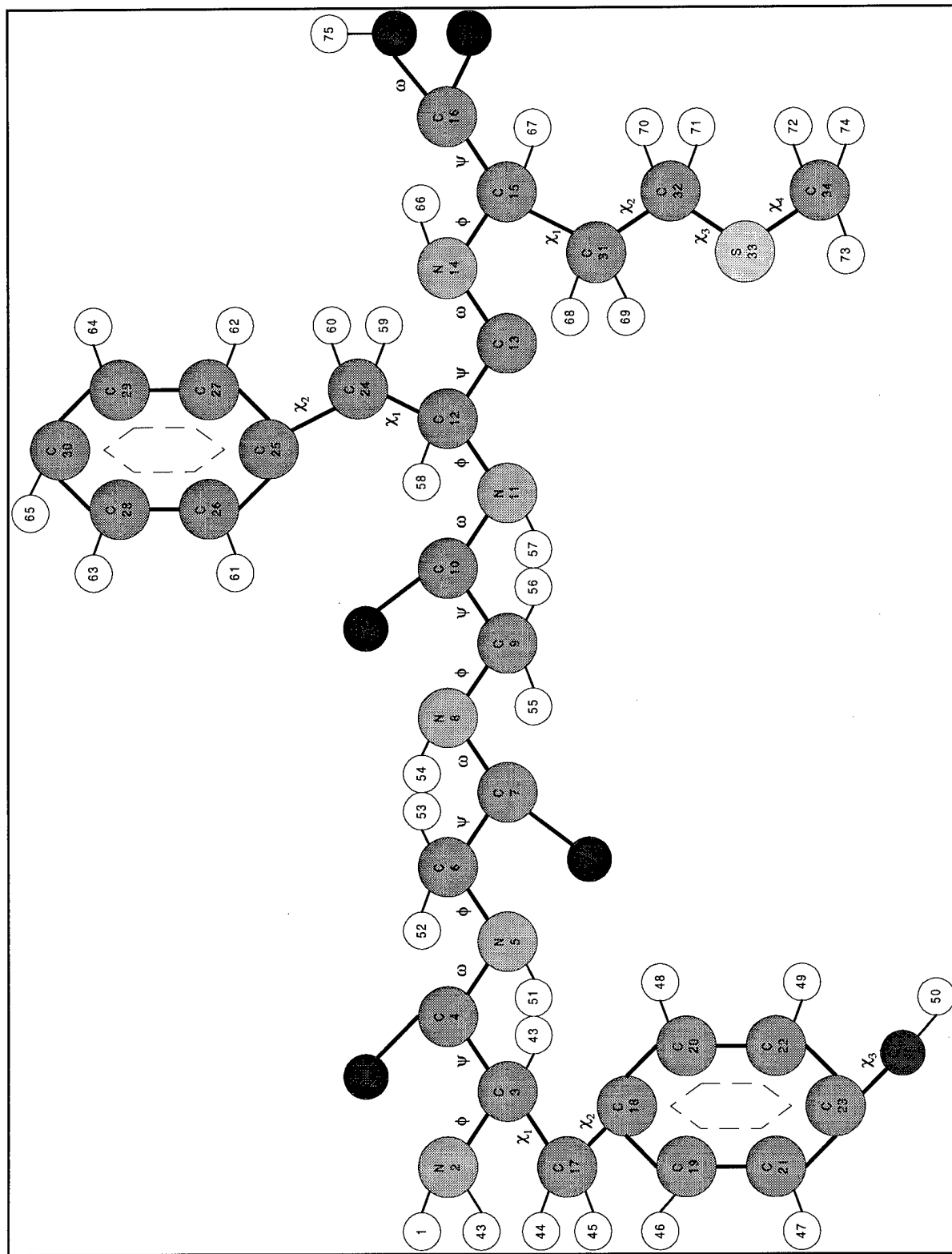


Figure 32: Extended Conformation of [Met]-Enkephalin

Residue	Dihedral			Angle (degrees)	
	Φ	Ψ	ω	χ_1	
Ala ₁	65°	(30° - 35°)	180°	60 -60 120	
Ala ₂	65°	(30° - 35°)	180°	60 -60 120	
Ala ₃	65°	(30° - 35°)	180°	60 -60 120	
⋮	⋮	⋮	⋮	⋮	
Ala ₅₇	65°	(30° - 35°)	180°	60 -60 120	

Table 15: Dihedral Angles for Polyalanine Accepted Energy Minimum

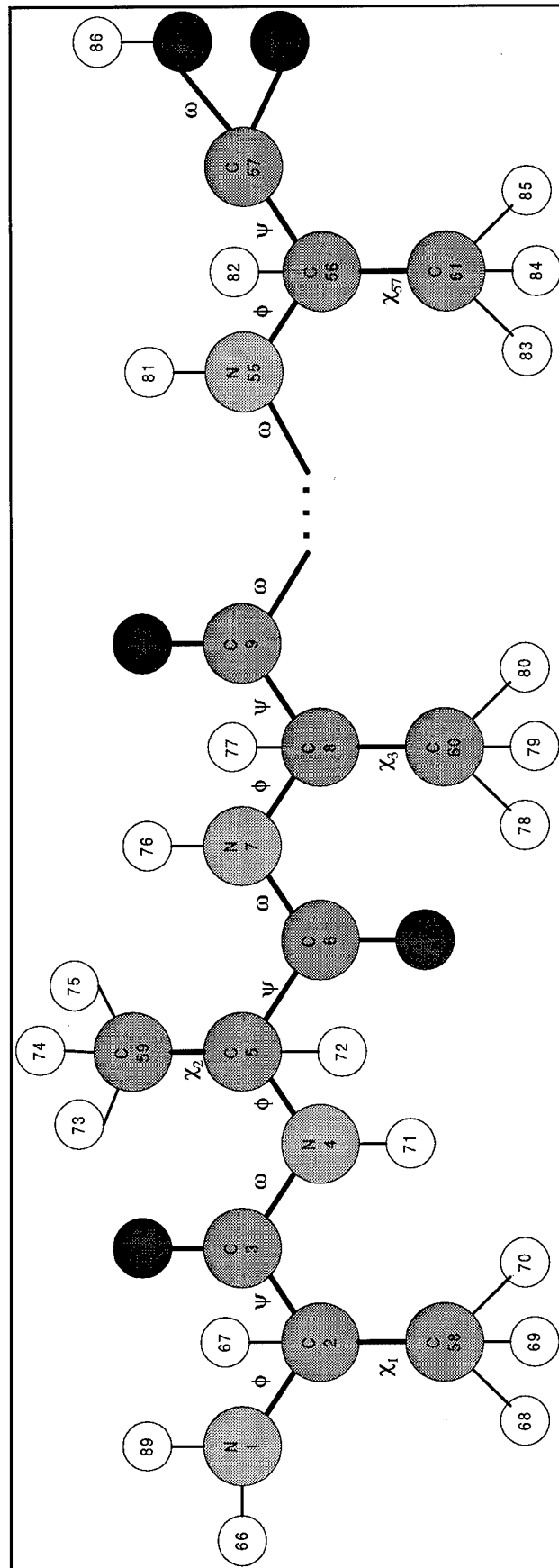


Figure 33: Extended Conformation of Polyalanine

5.2 General Data Requirements

Here we examine the general variety of data to be collected during the experiments, the random number seeds used to initialize the experiments, and discuss the general types of statistical tests that can be performed upon the collected data. Experiments are designed either to test effectiveness of a “new” algorithmic approach to a problem and/or to test the efficiency the algorithm has towards solving the problem. Our experiments examine both effectiveness and efficiency performance of the LLGA family.

5.2.1 General Data Requirements

Table 16 indicates the general types of data collected for each experiment. If a specific experiment requires additional data to be collected, the *methodology* portion of the experiment description identifies the additional requirements. The general types of data collected per evaluation include:

- ♦ The random number seed used,
- ♦ The best individual’s fitness value,
- ♦ The best individual’s chromosome (binary) representation,
- ♦ The best individual’s coordinates,
- ♦ The worst individual’s fitness value,
- ♦ The average fitness per generation,
- ♦ All the chromosomes evaluated and their fitness values,
- ♦ Any parameters,
- ♦ The overall execution time,
- ♦ The CHARMm execution time,
- ♦ The average CHARMm execution time, and
- ♦ The number of fitness evaluations performed.

Data Category	Description	Purpose	Experimental Purpose
Random Number Seed	The random numbers used to initialize each GA.	GAs are totally dependent upon the random number used to initiate the population.	Effectiveness
Best Individual's Fitness	Fitness of the best individual from each experiment.	This is one indication of the performance of a GA.	Effectiveness
Best Individual's Chromosome	The chromosome of the best individual from each experiment.	The chromosome indicates the individual's location within the search area.	Visualization
Best Individual's Coordinates	The coordinates of the best individual from each experiment.	The coordinates of each of the best individuals will be analyzed using RMS to determine the accepted energy minimum configuration.	Effectiveness
Worst Individual's Fitness	Fitness of the worst individual from each experiment.	This is one indication of the performance of a GA.	Effectiveness
Average Fitness	The average fitness of the population from each experiment.	An indication of the overall performance of the GA.	Effectiveness
All Chromosomes Evaluated & their Fitness Value	The chromosome each individual from each experiment.	The totality of chromosomes indicate where the GA was investigating.	Visualization
Parameters	All parameters manipulated within the GA.	Supplies all the information required to regenerate each experiment.	Historical Information
Execution Time	The total time for each experiment.	Used to indicate scalability and efficiency of the approaches.	Efficiency
CHARMm Time	The total time spent conducting fitness calculations.	Used to indicate scalability and efficiency of the approaches.	Efficiency
Average CHARMm Time	The average Time spent conducting fitness evaluations.	Used to indicate scalability and efficiency of the approaches.	Efficiency
The number of fitness evaluations.	The number of times the CHARMm is called.	A commonly used indication of efficiency.	Efficiency

Table 16: General Data Requirements for Each Experiment

5.2.2 Random Number Seeds

The LLGA requires the random seed to be a real number (of finite decimal representation) in the interval between 0 to 1. Since the LLGA bounds the random seed, we have chosen to use seven uniformly distributed random numbers within this range. The random seeds were pulled from [77] and scaled by placement of a decimal point prior to the first digit. **Table 17** indicates the seeds used for each GA experiment.

Set	LLGA (all variants)
1	0.014150
2	0.107629
3	0.241816
4	0.445874
5	0.680467
6	0.805648
7	0.928304

Table 17: Random Number Seeds Use In Initialization

5.2.3 Root Mean Squared Deviation (RMSD)

Root Mean Squared Deviation (RMSD; $p=2$ norm) is one way of comparing a calculated molecule's conformation to the naturally occurring conformation independent of the calculated fitness⁶². For instance, a GA may produce a molecule conformation (X), which approximates the accepted energy level of the best known conformation (Y), but internal coordinates for each atom within X may not closely correspond to the positions within Y. Thus, this analysis compares a GA best produced molecule to the naturally occurring molecule. RMSD is used in particular because the PSP community commonly references RMSD calculations. Therefore, we can compare our results to other research efforts, although, it must be stressed that different energy models may provide different energy values for the same molecule even when the internal geometry is the same. The general equation for this calculation is:

$$RMSD = \sqrt{\sum_{i=1}^{\text{number of dihedral angles}} (dihedral_{known} - dihedral_{calculated})^2}$$

Equation 20: RMSD Calculation

⁶² Other norms include: p -norm, maximum distance between points, maximum difference in probability, absolute difference, etc..

5.2.4 Test Platforms

Experiments are executed on the Aeronautical System Center (ASC) Major Shared Resource Center (MSRC) IBM SP2; the Air Force Institute of Technology (AFIT) network of workstations (NOWs); and on the AFIT ABC Beowulf. After initial testing, it was decided not to use the NOW due to its lack of computational power. A single execution of the LLGA using all available processors (6) requires 48 hours to complete⁶³. Table 18 indicates the number of processors used, the random number used, the population size, and the chromosomes per processor. Hardware limitations dictate the current upper limit for the AFIT Beowulf system. For a complete description of each system see **APPENDIX K**.

Number of Processors	Random Number	Population Size	Chromosome Per Processors	IBM SP2	AFIT Beowulf
1	0.014150	50	50	X	X
2	0.107629	50	50	X	X
3	0.241816	50	25	X	X
5	0.445874	52	13	X	X
9	0.680467	56	7	X	X
11	0.805648	50	5	X	X
17	0.928304	64	4	X	
25	0.999999	75 50	3 2	X	

Table 18: Processor Allocation per Architecture

5.3 Experiment Specifics

General parameters for the LLGA are given in the following table⁶⁴:

⁶³ See Appendix K for a complete description of each platform.

⁶⁴ **IMPORTANT!** See Appendix F for a sample LLGA input file, and the CHARMM energy model input files!

Parameter	Description	Impact Upon GA
building_blocks	Specifies the number of building blocks. [Met]-Enkephalin = 24 Polyalanine = 228	NO
(test function)	This field requires 5 entries. Building Blocks: 1- building_blocks Test Function: CHARMM_EVAL Bits per Building Block: 10 for both molecules Signal Ratio: 0.07 Weighting: 0	NO
coding_genes	Specifies the number of coding genes. It is the same as the chromosome length in the simple GA. [Met]-Enkephalin = 240 Polyalanine = 2,280	NO
noncoding_genes	The number of introns to include based upon Equation 43. [Met]-Enkephalin = 4,650 crossover disruptive 5% Polyalanine = 7,752 crossover disruptive 15%	YES
Popsiz	Size of the population. For the purposes of these tests we have fixed the population size to 50 chromosomes.	NO
selection_operator	<i>tournament selection between parents and children.</i>	YES
selection_rate	The number of chromosomes per tournament.	YES
Pcross	Ranges between 0 (never crossover) to 1 (always crossover).	YES
stop_criteria	Number of generation to execute the LLGA. With pcross = 1, the number of fitness evaluations per generation equals the population size.	YES
Seed	The random number generator seed.	YES
report_population	Write output to a file (on = yes, off = no).	NO
report_best_individual	Write output to a file (on = yes, off = no).	NO
Bbtemplate_filename	The filename of a file with a single chromosome in canonical form.	NO

Table 19: General LLGA Parameters

5.3.1 Experiment 1: Parallel vs. Sequential LLGA

Objective: The objective of this experiment is twofold. The first objective is to validate that results from the sequential LLGA (sLLGA) are equivalent to those obtained in the parallel (pLLGA) version. Many man-hours have been spent ensuring the behavior of the pLLGA is in accordance with the sLLGA. Furthermore, correctness is an issue because there is greater difficulty in verifying correctness of parallel algorithms

than sequential algorithms [36, 37]. Secondly, we wish to characterize the efficiency of the pLLGA in terms of overhead, speed-up, and scalability.

Methodology: Harik's serial implementation [17] which we modified to incorporate the CHARMM energy model is used for the sLLGA.

A data parallelized implementation based upon the sequential LLGA is employed for the multi-node experiments. **Chapter 4** presents low-level design and implementation details. Comparison between the single node and the 2 node parallel runs are used to calculate the empirical overhead imposed by the communications library. Parallel executions using 2, 8, 10, 16, and 25 processors are used to evaluate scalability and speed-up of this algorithm.

Parameters for each test case are supplied in **Table 20**. Since the purpose of this test is to characterize the two implementations, a single random seed for all experiments is selected. The random seed corresponds to set 3 from **Table 17**.

Results and analysis are presented in (**SECTION 6.1**).

Parameter	Description
selection_operator	<i>tournament selection with replacement</i> <i>tournament selection w/out replacement.</i>
selection_rate	4
pcross	0.70
stop_criteria	1,000 generations
seed	0.241816
report_population	on
report_best_individual	on
BBtemplate_filename	BBtemplate.txt

Table 20: Parameters for Experiment 1

5.3.2 Experiment 2: Constrained vs. Non-constrained Sequential LLGA

Objective: **Chapter 2** indicates a growing body of information being developed about the PSP problem domain. Intuitively, it seems that if we incorporated new information in order to constrain our search space, then perhaps we could improve search performance. Charles Kaiser pioneered AFIT's expedition into this realm by incorporating the constraints developed by Ramachandran [61]. His experiments showed great promise, but he encountered some computational problems (**see SECTION 2.4**). As discussed in **SECTION 2.5**, we have re-evaluated and re-implemented Kaiser's work directly into the decoding of the binary chromosome. The

validation of this new design and implementation is the objective of this experiment. The two implementations are compared for effectiveness and efficiency.

Methodology: Experiments for each test case are executed as in experiment 1. The “best” molecule found by both implementations is compared based upon total energy and RMSD from the accepted conformation. Furthermore, since constraining the LLGA impacts the algorithm’s execution rate, the execution time for each implementation is compared to determine the cost of additional calculations.

Low-level implementation details for the constrained-LLGA are given in **CHAPTER 4**. Parameters for each test case are supplied in **Table 21**. The results from this experiment are summarized in (**SECTION 6.2**).

Parameter	Description
selection_operator	<i>tournament selection with replacement</i> <i>tournament selection w/out replacement.</i>
selection_rate	4
pcross	0.70, 0.75, 0.80, 0.90
stop_criteria	1,000 generations
seed	The seeds used follow Table 17 .
report_population	on
report_best_individual	on
BBtemplate_filename	BBtemplate.txt contains the Quanta conformation.

Table 21: Parameters for Experiment 2

5.3.3 Experiment 3: Constrained Parallel LLGA vs. Non-constrained Parallel LLGA

Objective: The basis for this experiment is to characterize the efficiency of a pLLGA that also incorporates the insight gained by the constrained-LLGA. By parallelizing the constrained-LLGA, we may receive even larger efficiency dividends than from the sequential version.

Methodology: Experiments for each test case are executed as in **Experiment 1**. A data parallelized design and implementation based upon the sequential constrained-LLGA is used for the multi-node experiments. **Chapter 4** gives the low-level design and implementation details. Comparison between the single node and the 2 node parallel runs are used to calculate the overhead imposed by the communications library as in experiment 1. Parallel executions using 2, 8, 10, 16, and 25 processors are used to evaluate scalability and speed-up of this algorithm. RMSD calculations are performed on the best resulting chromosome.

The implementation of the constrained parallel LLGA is a coupling between the constrained-LLGA and the pLLGA. Design and implementation details for each are given in **CHAPTER 4**. Parameters for **Experiment 3** are the same as those presented in **Experiment 2**. Results are presented in (**SECTION 6.3**).

5.3.4 Experiment 4: Constrained-pLLGA vs. Constrained Para-REGAL Implementation

Objective: The objective of this experiment is to characterize the constrained-LLGA against previous AFIT GA/PSP implementations. As stated earlier, Kaiser [37] pioneered AFIT's approach to constraining the search space of the PSP problem, and as discussed in **CHAPTER 2**, he encountered a few shortcomings with his implementation. On the other hand, he was able to uncover the "lowest" known [Met]-Enkephalin energy to date (-30.32 kcal/mol) [52]!

Methodology: Experiments for each test case are executed using 11 processors on the Air Force Institute of Technology (AFIT) ABC Beowulf.

The best chromosome found is compared with Kaiser's results and to the native conformation using RMSD. The Quanta defined conformation is used as the template for successive testing using the same random seed. Efficiency of the two algorithms is also characterized even though Kaiser's implementation is serial. Results are presented in (**SECTION 6.4**). Parameters used are in **Table 22**.

Parameter	Description
selection_operator	<i>tournament selection with replacement</i> <i>tournament selection w/out replacement.</i>
selection_rate	4
pcross	The best rate as determined by earlier test.
stop_criteria	1,000, 10,000, and 20,000 generations
seed	The seeds used follow Table 17 .
report_population	on
report_best_individual	on
BBtemplate_filename	BBtemplate.txt. With each successive execution the best molecule found is fed in as the next BB template.

Table 22: Parameters for Experiment 4

5.3.5 Experiment 5: Constrained-LLGA, Non-constrained LLGA, pLLGA, constrained pLLGA vs. fmGA

Objective: Previously, the most promising Linkage Learning GA (LIGA) employed is the fmGA. This test compares the results of the fmGA to the LLGA family. It is anticipated that the LLGA family of GAs grossly outperforms the fmGA.

Methodology: Experiments for each test case are run using 1, 2, 8, 10, 16, and 25 processors on the SP2, and each experiment is executed using 1, 3, 9, and 11 processors on the ABC Beowulf. Harik's serial implementation [17], which is modified to incorporate the CHARMm energy model, is used for the sLLGA.

The data parallelized implementations are used for the multi-node experiments. **Chapter 4** gives the design and implementation details. Comparison between the single node and the 2 node parallel runs are used to calculate the overhead imposed by the communications library. Parallel executions using more than 2 processors are used to evaluate scalability and speed-up of the different implementations. (Note: twelve processors are the current upper limit upon the ABC Beowulf due to hardware constraints.)

The best molecule found in each run is used in as the next execution template. The absolute best molecules uncovered undergo RMSD evaluation against the accepted conformation. Results are presented in (**SECTION 6.5**).

Parameters for each test case are supplied in **Table 23**.

Parameter	Description
selection_operator	<i>tournament selection with replacement</i> <i>tournament selection w/out replacement.</i>
selection_rate	4
pcross	The best rate as determined by earlier test. The crossover rate in the fmGA is set to 0.70 with mutation set as 0.01.
stop_criteria	1,000 generations
seed	The seeds used follow Table 17 .
report_population	on
report_best_individual	on
BBtemplate_filename	BBtemplate.txt. With each successive execution the best molecule found is fed in as the next BB template.

Table 23: Parameters for Experiment 5

5.4 Summary

The methodology outlined in this chapter is used to analyze the LLGA, the parallel LLGA, and the constrained LLGA against previous AFIT PSP algorithm

implementations. The objective and parameters for each experiment are laid out as well as the basis for validating the results. Finally, we note that the data from AFIT's previous experiments is used; we are not going to re-execute past research.

6.0 Results and Analysis

The results from the experiments put forth in **Chapter 5** are summarized in the following sections. Raw data is available in electronic format. Each experiment was executed as documented. Furthermore, some general observations concerning the execution behavior of the LLGA are documented in **Section 6.7**. Our visualization of the [Met]-Enkephalin energy landscape is discussed and analyzed in **Section 6.8**.

6.1 Experiment 1: Parallel vs. Sequential LLGA

The first objective of this test is to validate the results from the sequential LLGA are equivalent to those obtained by the parallel LLGA (pLLGA). Since the random seed were the same during all executions, comparing the “equivalence” between the two implementations is little more than ensuring that the same end results are produced by both implementations. **Table 24** indicates that the energy characteristics for the LLGA and the pLLGA are the same. This comparison alone does not irrefutably conclude that both Linkage Investigating Genetic Algorithms (LIGAs) behave identically because the ruggedness of the energy landscape may skew the results.

Implementation	Optimal Energy Found	Average Population Energy	Worst Energy
LLGA	-9.86312	0.351708	18.9437
pLLGA	-9.86312	0.351708	18.9437

Table 24: End of Execution Energy Comparison between LLGA and pLLGA

It is possible that the [Met]-Enkephalin energy landscape searched by both implementations could have similar energy characteristics. Therefore, to further prove that both implementations behave similarly, we compared the final populations from the two executions. This comparison showed that both LIGAs produced the identical final populations. Therefore, we can conclude that both LIGA implementations search identical areas of the protein's energy landscape.

Secondly, the efficiency of the pLLGA in terms of communication overhead, speedup, and scalability was characterized. **Table 25** lists the average execution times for the pLLGA running on the ABC Beowulf and Maui High Performance Computing Center's (mHPCC) SP2, respectively.

Number of Processors	Average Execution Time (seconds)	Average Execution Time (hours)
1	57123.75	15.87
2	50839.64	14.12
3	36970.50	10.27
5	19677.14	5.47
9	15735.44	4.37
11	18085.14	5.02

Table 25: pLLGA Average Execution Times - ABC Beowulf

Number of Processors	Average Execution Time (seconds)	Average Execution Time (hours)
1	85403.22	23.72
2	87660.48	24.35
3	57451.06	15.96
5	40280.68	11.19
9	33578.82	9.33
11	29388.20	8.16
17	32562.04	9.05
26	24699.22	6.86

Table 26: pLLGA Average Execution Times - Maui SP2

The total overhead is the difference between the cost of performing the problem on a single processor and the cost of performing the same task on the parallel architecture. This "cost" represents the amount of time the parallel implementation consumes performing communications, which is the penalty for using a parallel application. **Equation 21** illustrates the communication overhead (T_o) where T_p is the parallel time and T_s is the best sequential time to complete the task [36]:

$$T_o = T_p - T_s$$

Equation 21: Total Overhead

By subtracting the single processor version from the two-processor version of the pLLGA executed on like processors, we calculated the total overhead for the pLLGA implementation as 0.57 hours on the AFIT Beowulf and 0.63 hours on the mHPCC SP2. These two implementations perform the same amount of work except the 2 processor pLLGA farms out its fitness calculations to a slave processor. Of course, the calculated overhead hours represent the total overhead accumulated over 5,000 generations. The

total overhead per generation is only 0.41 (Beowulf) and 0.45 seconds (SP2), which is much more reasonable when considering the amount of data being passed between the two processors.

Speedup (S) is a measure capturing the relative benefit of solving a problem in parallel. **Equation 22** defines the speedup calculation [36].

$$S = \frac{T_s}{T_p}$$

Equation 22: Speedup

There are two terms with which one must be familiar with when discussing speedup. The first is *linear speedup*. Linear speedup increases proportionally with the number of processors. Super-linear speedup, the second term, is when $S > p$ (p is the number of processors). Although, this phenomenon may be observed it is usually due to either 1) a non-optimal sequential program or 2) the parallel programs ability to take better advantage of the memory hierarchy [36]. **Figure 34** illustrates the speed-up obtained by parallelizing the LLGA over the range of possible ABC Beowulf and mHPCC SP2 processors.

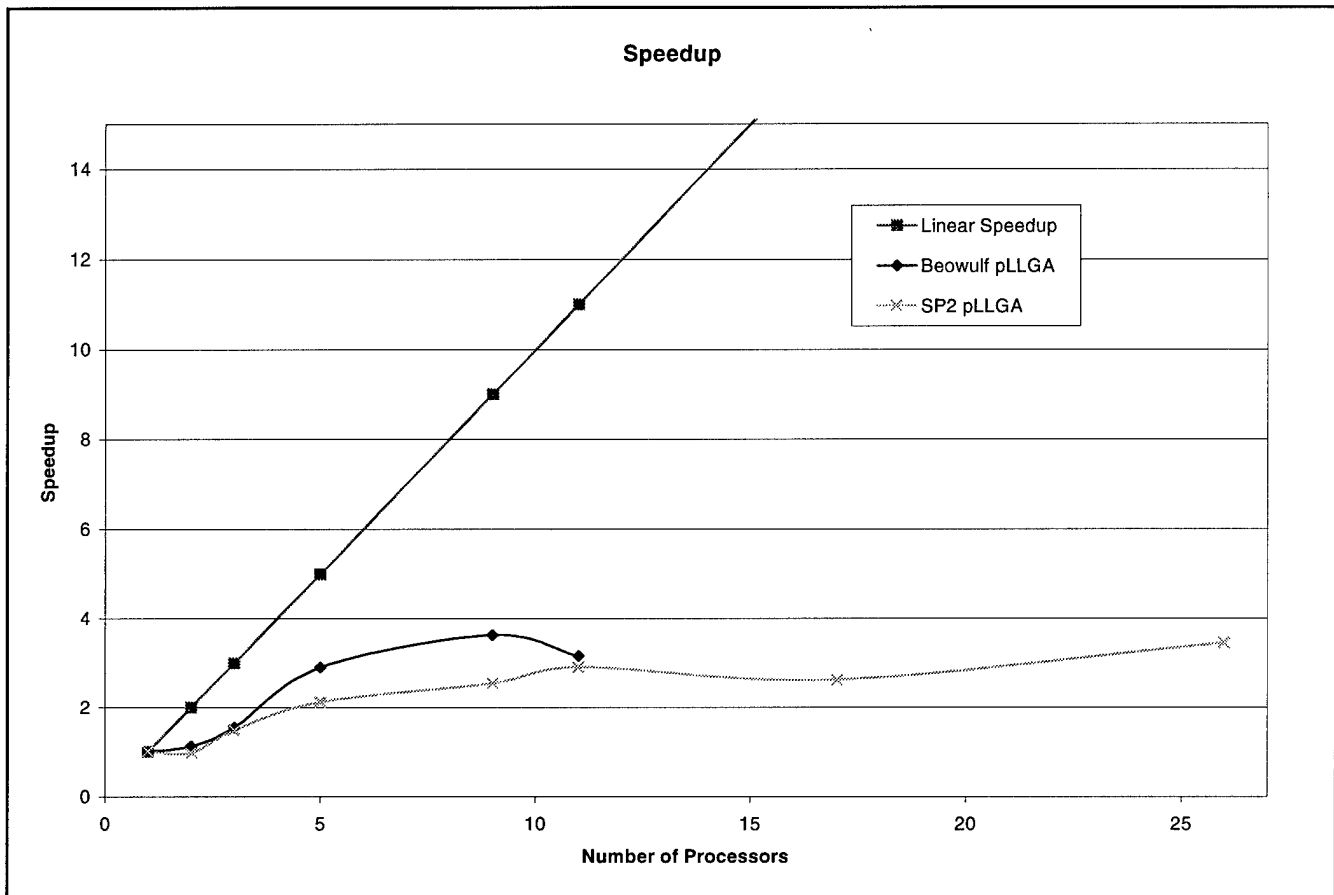


Figure 34: pLLGA Speedup

From **Figure 34**, the ABC Beowulf speedup curve indicates that once 5 processors have been applied to the task no more speedup is obtained. Actually, we see a decrease in the achieved speedup. What happens at the five processors point is that the ratio between computation and communication shifts from more computation time required to more communication time required. The steep decline from 9 processors to 11 processors shows the addition of the 200 MHz. Pentium Micron. This much slower processor hampers the computational performance of the implementation as well as the communications between the processors. The standard deviation and variance for **Figure 34** and **Figure 35** is provided in **Table 27**.

Number of Processors	Beowulf		SP2	
	Standard Deviation	Variance	Standard Deviation	Variance
1	36645	1342880866	74	5475
2	458	209768	6039	36463842
3	7664	58743450	4575	20931207
5	12	142	2565	6581009
9	84	4936	651	423461
11	58	3416	1708	2917504
17	—	—	397	157565
26	—	—	1498	2243089

Table 27: Statisticals For pLLGA Implementations

On the mHPCC's SP2, the pLLGA seems to achieve "some" speed-up each time we increased the number of processors. This is due to the optimized communications backbone of the SP2 that tilts the balance of computation to communication ratio towards the computation side of the equation. On the SP2, there are dedicated processors to handle the communications, but on AFIT's Beowulf there are no dedicated processors just for communication. Therefore, every time a message is passed additional communication overhead is generated on AFIT's Beowulf.

Finally, looking at the efficiency (E) of the pLLGA implementation, we see the same behavior. Efficiency, governed by **Equation 23**, is a measure of the amount of time for which a processor is accomplishing useful work (i.e., not idle) [36]. The efficiency of the pLLGA steadily increases until the 5-processor mark from which point it continues to drop off. Once again, this indicates that the communication overhead is beginning to dominate the parallel performance equation. The initial dip in the ABC Beowulf's performance indicates the move from 2 processor (the sequential application) to 3 processors. On the other hand, the efficiency of the pLLGA on the SP2 is horrible. The SP2/pLLGA combination reaches 50% efficiency at 3 processors then sharply drops to only 13% efficiency with 26 processors. This indicates that the pLLGA is not scalable on the SP2.

$$E = \frac{S}{p}$$

Equation 23: Efficiency

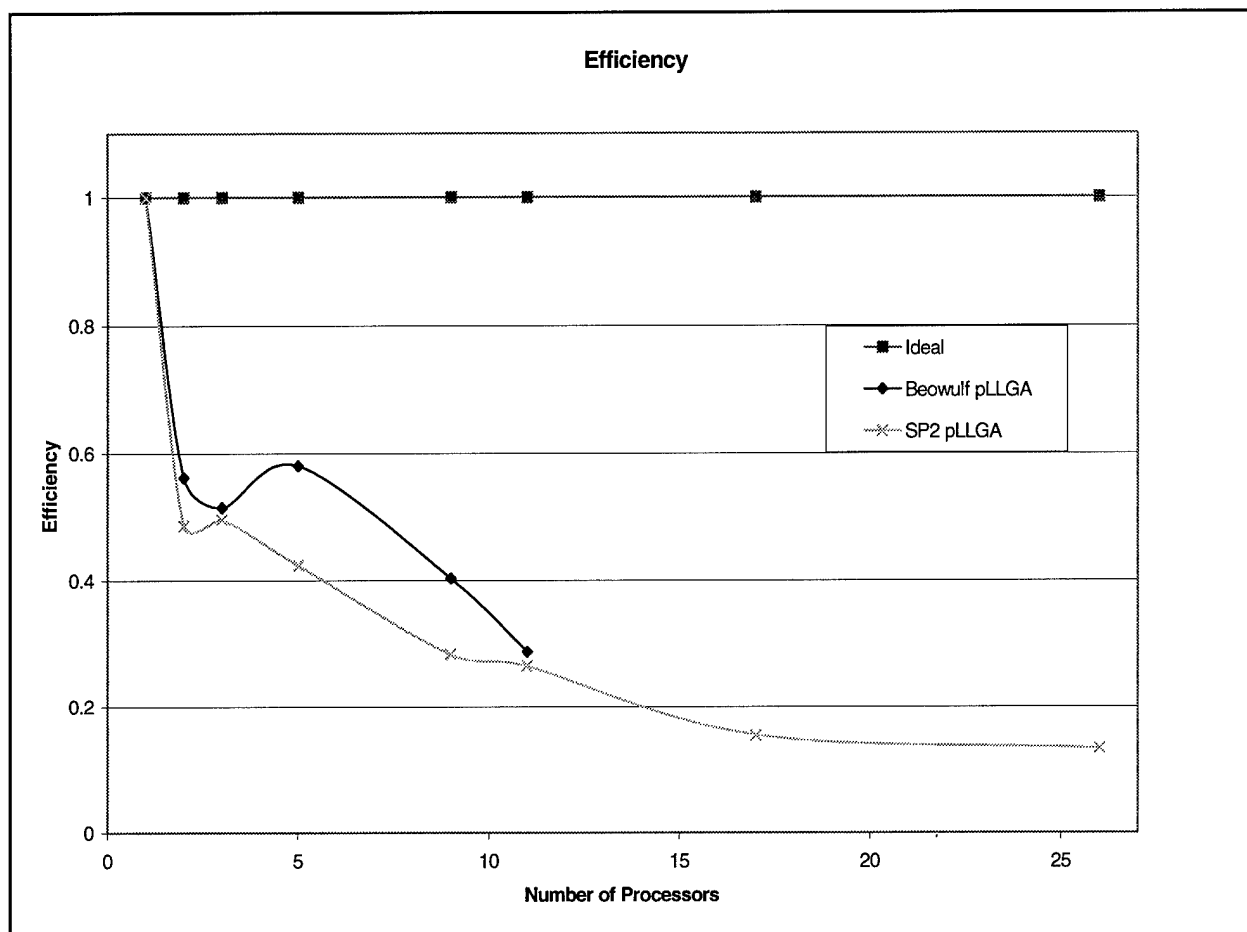


Figure 35: pLLGA Efficiency

6.3 Experiment 2: Constrained vs. Non-constrained Sequential LLGA

Our tests indicate that the inclusion of the constraints into the decoding of the chromosome add a negligible amount of overhead to the LLGA's execution time.

	ABC BEUWOLF		IBM SP2	
	cLLGA	LLGA	cLLGA	LLGA
Average Execution Time (sec)	40902	36343	89235	85403
Average Execution Time per Energy Calculation (sec)	0.1383	0.1213	0.2811	0.2638

Table 28: Constrained vs. Non-Constrained

Table 28 indicates that for both platforms the inclusion of the constraints into the decoding of the chromosome behaves as expected. Since the re-engineering of the AFIT's CHARMm energy model to include Ramachandran constraints meant including one additional add, subtract, multiply and divide operation per chromosome evaluated, it

was not expected that this “new” methodology would overwhelm the computational time of the algorithm.

On the other hand, the inclusion of the constraints had a noticeable affect on the effectiveness of the algorithm. **Figure 36** and **Figure 37** show the energy characteristics for the cLLGA and the LLGA, respectively. As can be seen in **Figure 36**, the cLLGA quickly narrows the breath of the search area as indicated by the sharp initial drop in the energy trend lines. This is due to the constraints put on the search space and is not an effect of a change to the LLGA algorithm. Finally, the final energies uncovered by the cLLGA are much better than the LLGA as indicated in **Table 29**.

Algorithm	Optimal Energy	Average Energy	Maximum Energy
cLLGA	-16.3584	-10.426	-1.87393
LLGa	-9.86312	0.351708	18.9487

Table 29: Final Energy Characteristics for the LLGa and cLLGA

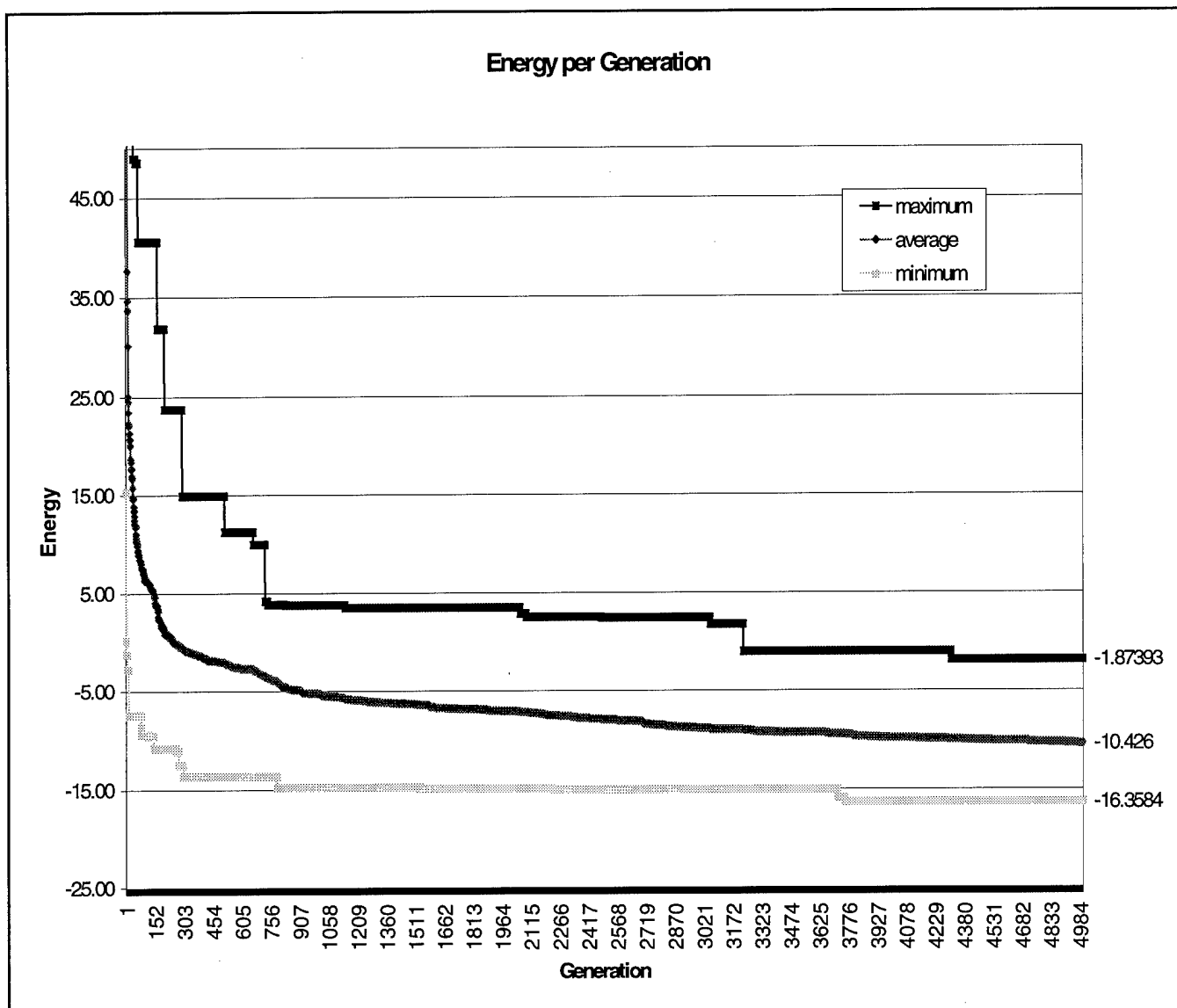


Figure 36: Energy Characteristics of the cLLGA

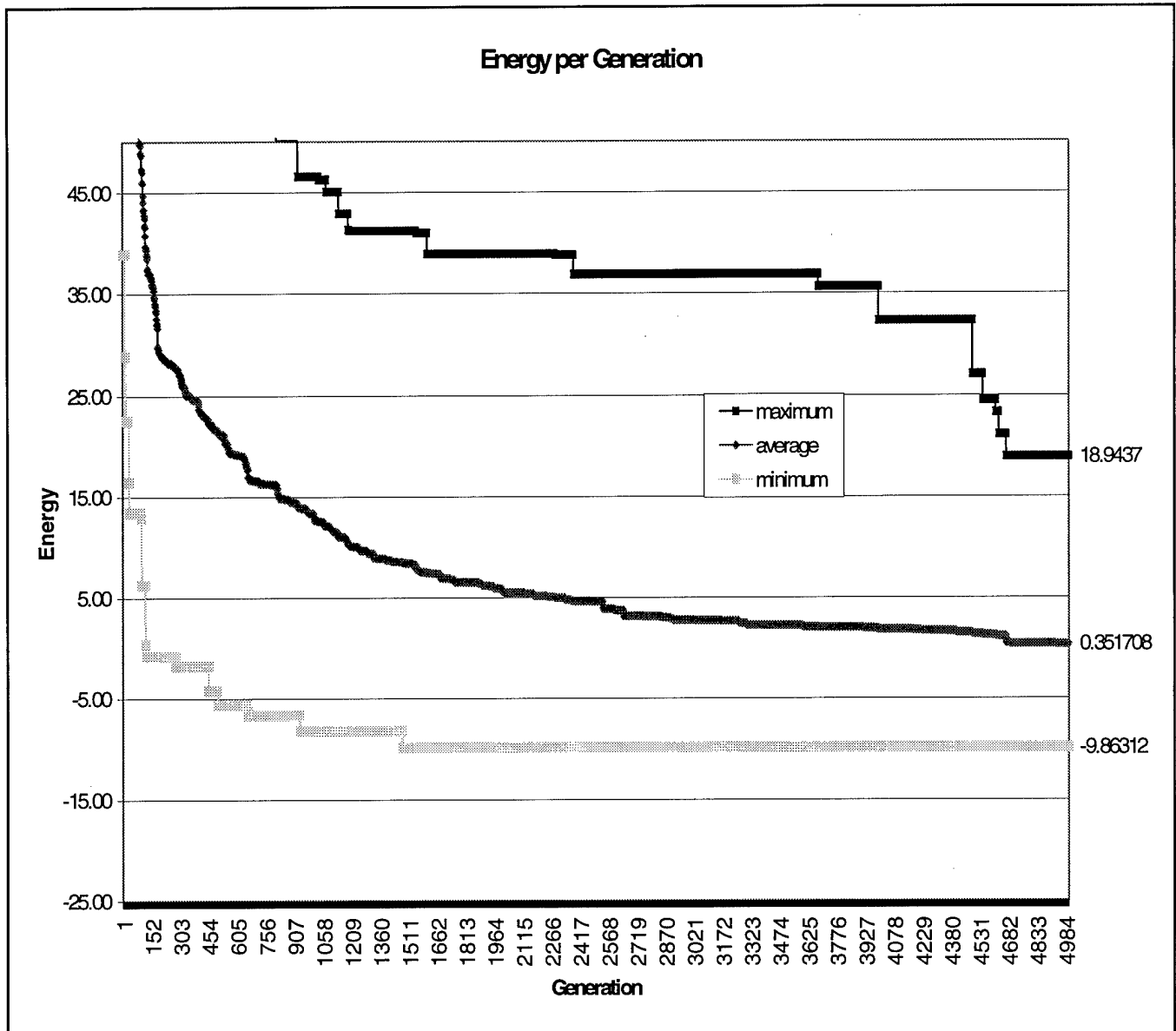


Figure 37: Energy Characteristics of the LLGA

6.4 Experiment 3: Constrained Parallel LLGA vs. Non-constrained Parallel LLGA

As stated earlier in **Section 6.3**, the additioned overhead added from the constraints did not noticeably effect the execution time of the LLGA. Therefore, the comparison between the overhead, speed-up, and efficiency of the constrained parallel LLGA (cpLLGA) and the non-constrained pLLGA does not reveal startling new information. **Table 30** compares the calculated overhead for these two

implementations. As expected, there is not much difference between both implementations' overhead.

Algorithm	Beowulf	mHPPC SP2
pLLGa	0.57 hours	0.63 hours
cpLLGA	2.31 hours	4.76 hours

Table 30: Total Overhead for cpLLGA and pLLGA

On the AFIT's Beowulf, the overhead per algorithm is 0.41 seconds (pLLGA) and 1.66 seconds (cpLLGA) per generation, whereas, on the mHPCC SP2 the overhead per generation per algorithm is 0.45 seconds and 3.42 seconds, respectively. The higher overhead is attributed to the differences in processor capability.

Figure 38 shows the speedup of the cpLLGA as compared to the pLLGA on both test platforms. As expected, the speedup of the cpLLGA is nearly identical to the pLLGA. There is a noticeable difference in the cpLLGA executing on AFIT's Beowulf with five or more processors. In this configuration, the cpLLGA's slave processors are required to perform additional computations. This shift in computation is reflected by the increase time spent in parallel operations that directly affect the speedup and efficiency calculations. The sharp decline in speedup for the cpLLGA is a result of adding the much slower 200 MHz. Pentium Micron.

A similar pattern is seen in **Figure 39**. **Figure 39** shows the efficiency of the two different algorithms. Again, the additional calculations of the cpLLGA makes this algorithm more efficient because now each slave processor is required to perform a greater share of the overall computation. Therefore, these processors are idle for less of time as compared to the slave processors in the pLLGA implementation.

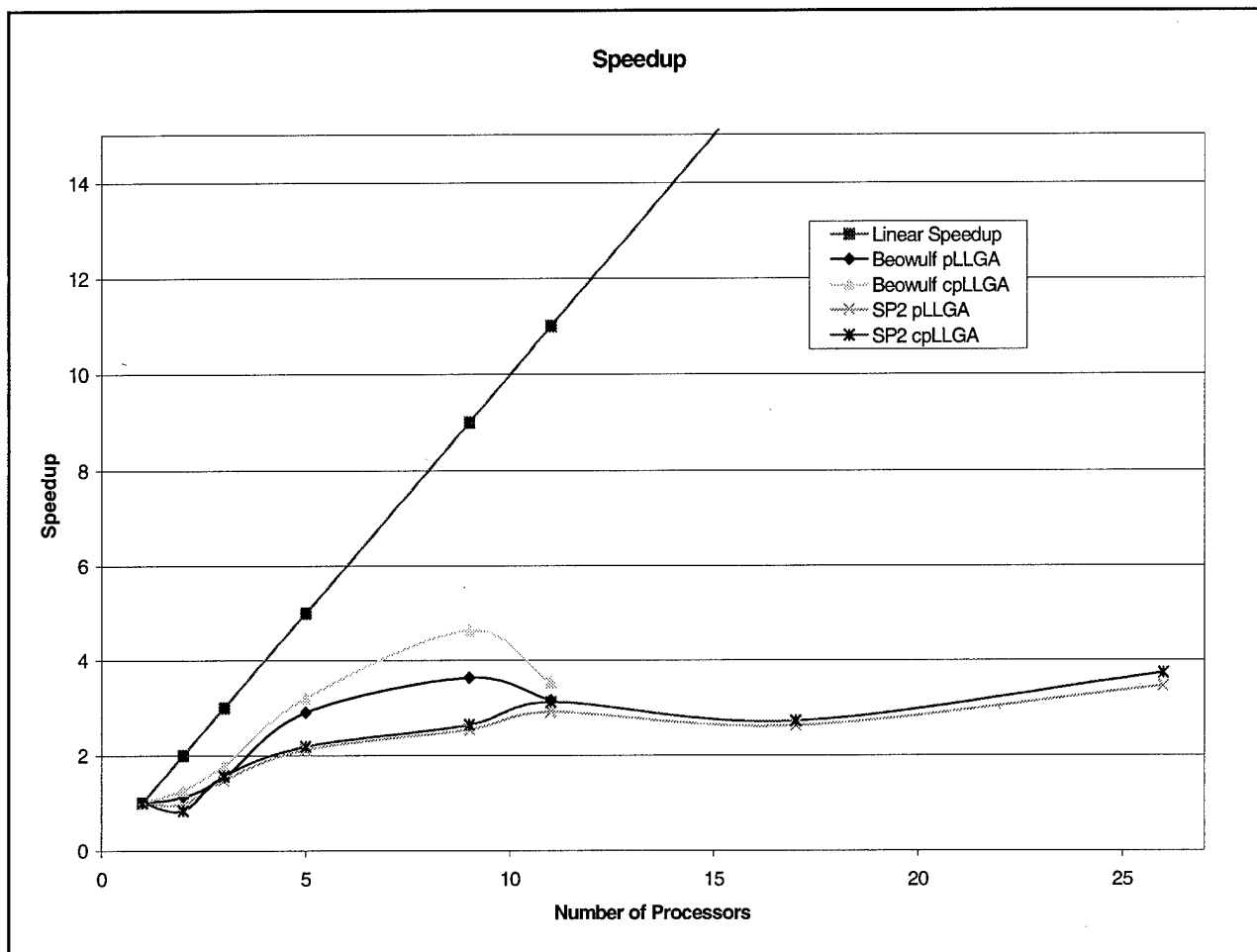


Figure 38: Speedup Comparison between pLLGA and cpLLGA

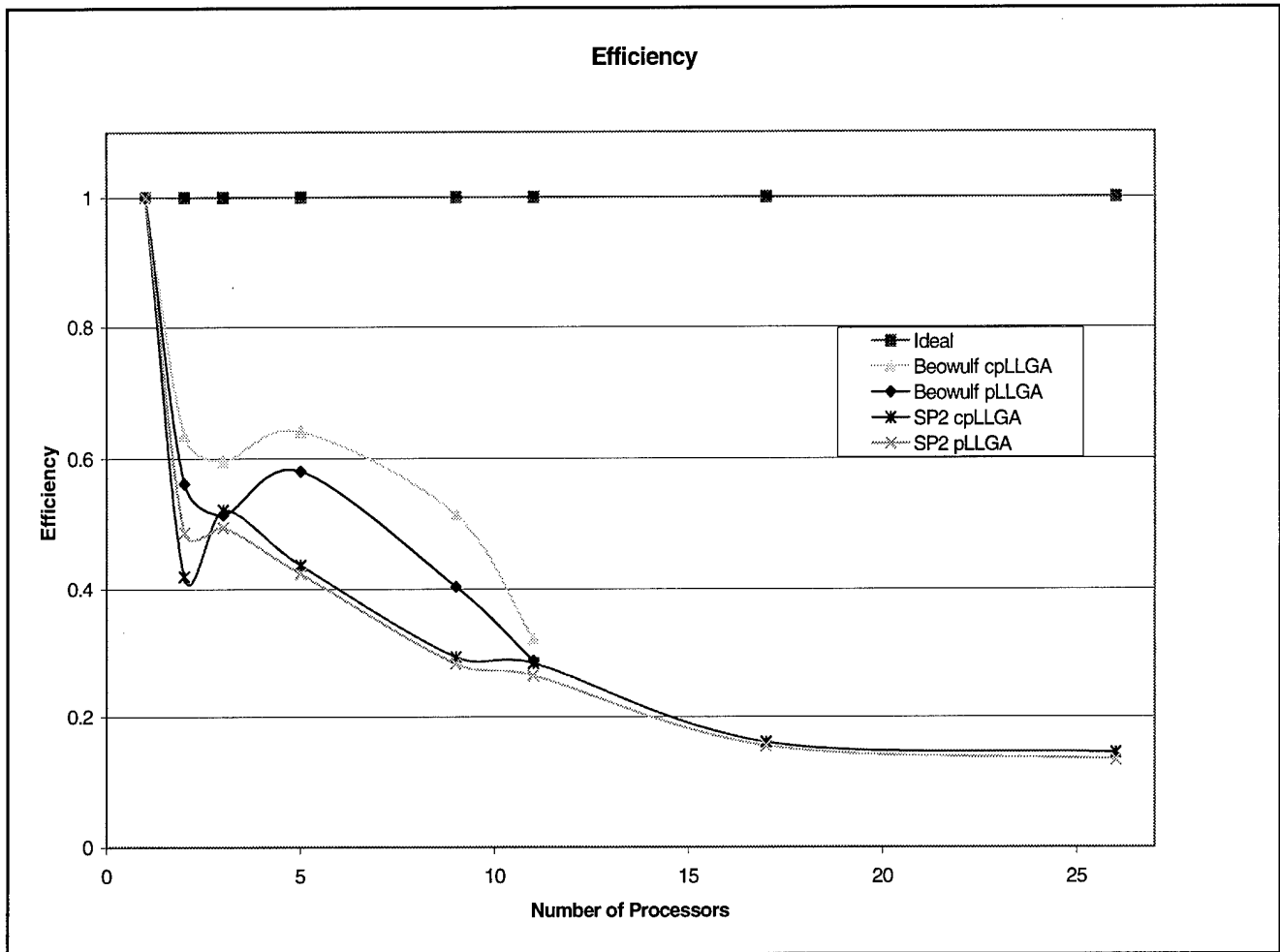


Figure 39: Efficiency Comparison between the pLLGA and cpLLGA

6.5 Experiment 4: Constrained-pLLGA vs. Constrained Para-REGAL Implementation

Although Kaiser does not provide any efficiency evaluation for his Para-REGAL system, he does supply enough data to piece together a rough comparison between our respective approaches. Kaiser executed his experiments for 100,000 evaluations using a population size of 50 on four separate islands [37]⁶⁵. On average, these tests expended 4.675 hours [37]. The constrained-LLGA (serial version) used a population size of 50 and terminated with 250,051 evaluations and consumed 11.362 hours on average. Therefore, the constrained-LLGA accomplished 2.5 times the amount of work

⁶⁵ There is no data indicating the number of processors used. We have assumed he used 1 processor per island for a total of four processors.

in approximately 2.43 times the amount of time. Thus, constrained-LLGA is slightly more efficient than the Para-REGAL system. Expanding this comparison to include the constrained-pLLGA using only three processors⁶⁶, the constrained-pLLGA outperforms Kaiser's Para-REGAL system. This implementation using fewer processors accomplished more than twice the amount of work (250,051 evaluations) in less than twice the amount of time (8.195 hours).

These results could be skewed towards the constrained-pLLGA because, although Kaiser does not explicitly state the system architecture used to evaluate his Para-REGAL system, the best available systems could have been the Ultra Sparc Workstation Network. We excluded this system from our test platforms because of its much smaller computational power in comparison to the IBM SP2 or AFIT ABC Beowulf (see **Chapter 5**). Our initial testing indicated that the Ultra Sparc Workstation network ran nearly twice as long to execute the same application as the ABC Beowulf. Taking this into account greatly closes the gap between the efficiencies of these two separate approaches.

6.6 Experiment 5: Constrained-LLGA, Non-constrained LLGA, pLLGA, constrained pLLGA vs. pfmGA

As seen in **Figure 40**, the pfmGA outperforms every member of the LLGA family, but this was not expected from the algorithmic discussion in Chapter 3. The LLGAs were an order of magnitude less complex than the fmGA. The better overall execution times for the pfmGA can be explained by better parallelism. Gates pfmGA demonstrated super linear speedup using 2, 4, and 8 processors [15]. Furthermore, even when his algorithm was rated as less than linear speedup (+16 processors), Gates' pfmGA implementation was still achieving 9-fold speedup. The LLGA never achieved above 4-fold speedup on the ABC Beowulf. The LLGA family suffers from a very closely matched communications to computations ratio. Therefore, parallelization of this algorithm does not achieve the anticipated dividends.

From an effectiveness standpoint, again the LLGA family is grossly outperformed. The most optimal solution generated by the LLGA implementations had a conformation energy of -18.22 kcal/mole and an RMSD of 17.124 this protein was uncovered by the pLLGA using 0.241816 as the random seed. The RMSD calculation

⁶⁶ One master and only two slaves.

was performed against the “optimal” QUANTATM [Met]-Enkephalin molecule discussed in **Section 5.1**.

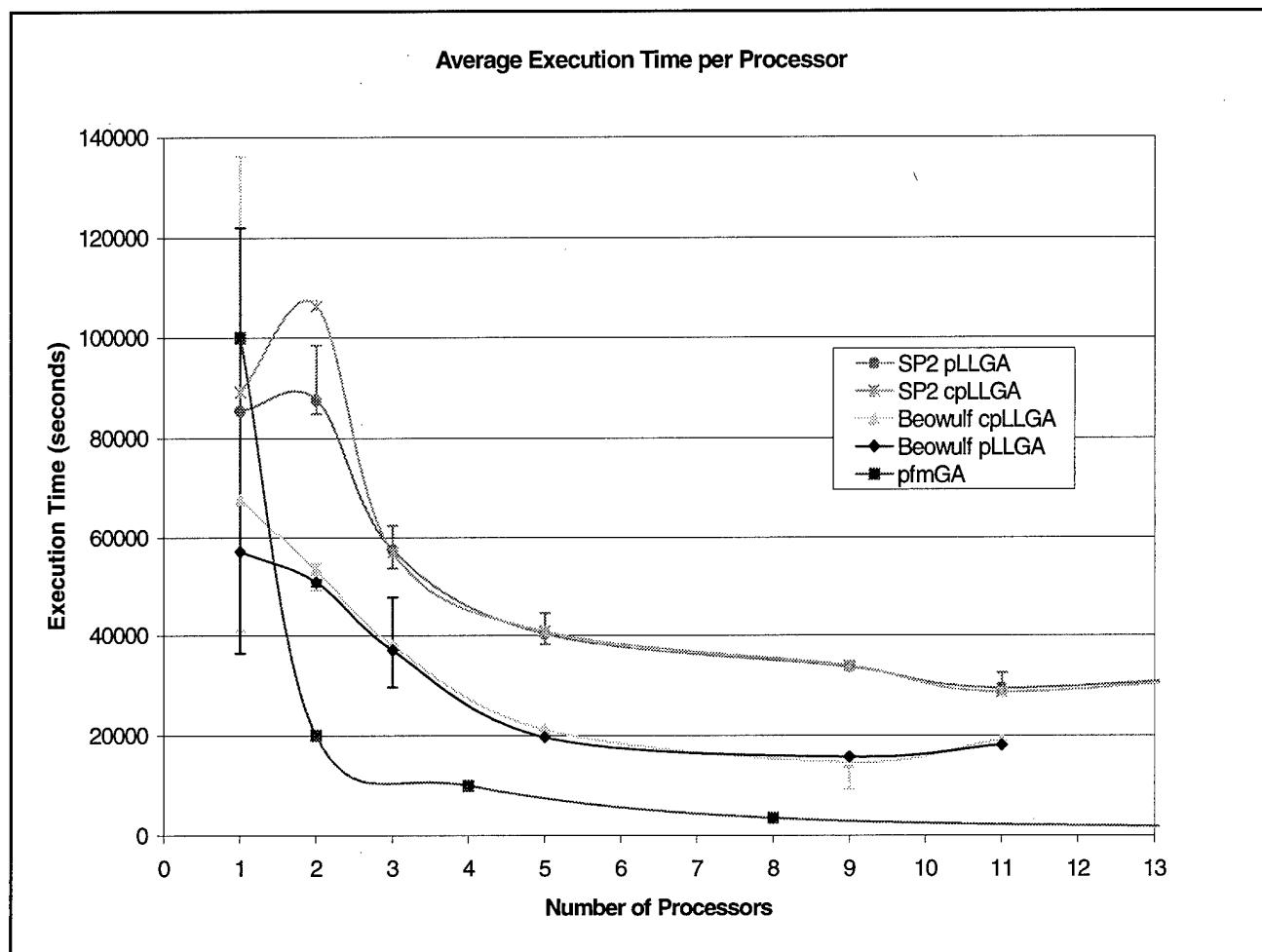


Figure 40: Comparison Between Linkage Investigating Gas

6.6 LLGA Observations

Additional observations were made concerning the LLGA's performance that did not correspond to any of the test cases documented in Chapter 5. Furthermore, these “observations” were made over the course of the five tests and are considered as general conclusions concerning the performance of the LLGA implementations.

The first major observation that concerned us greatly is the LLGA's inability to maintain building blocks (BBs) once they are within the population. The following figures represent the average number of BBS contained within the population per generation for the pLLGA.

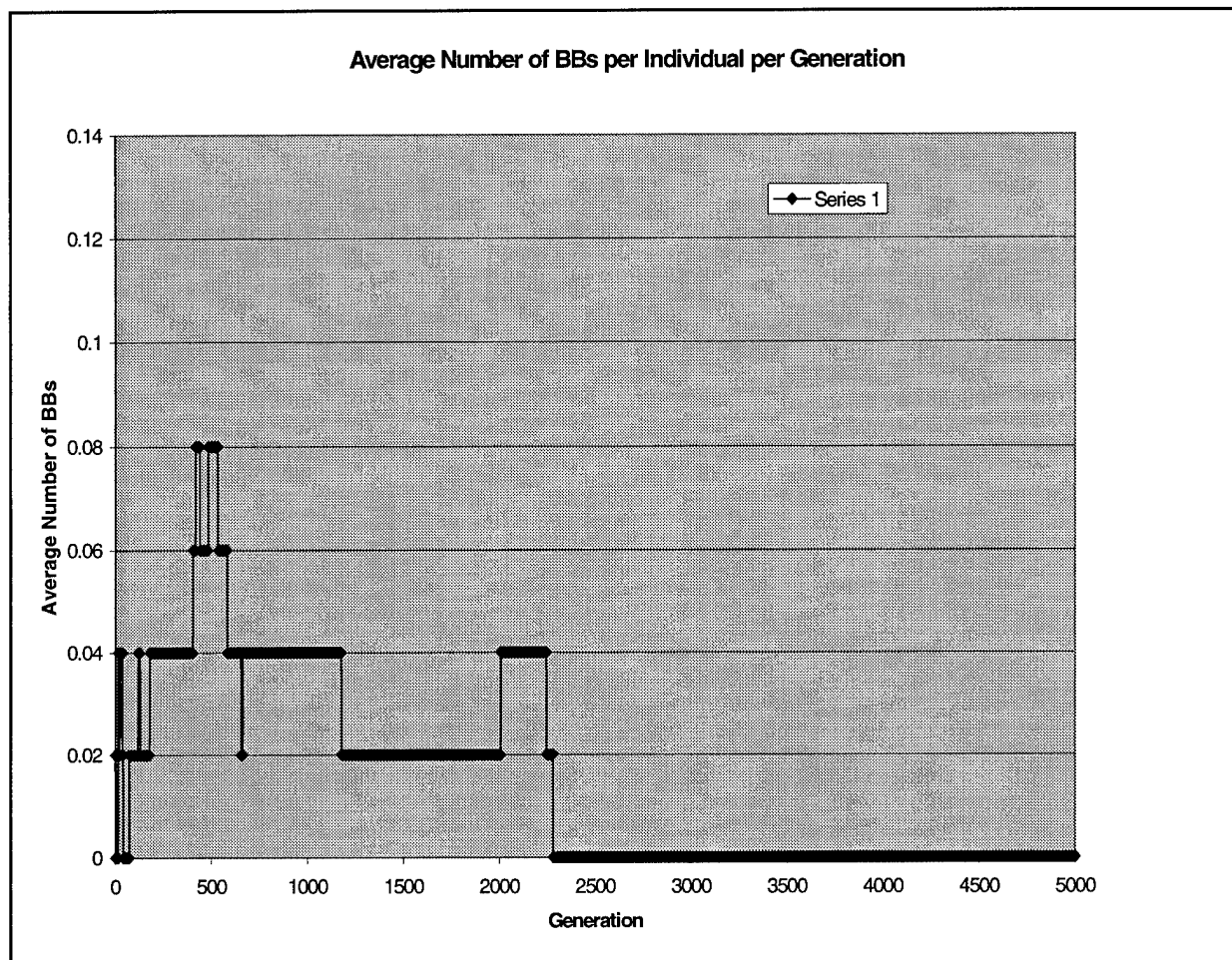


Figure 41: BBs Uncovered and Maintained for the pLLGA and Random Seed 1

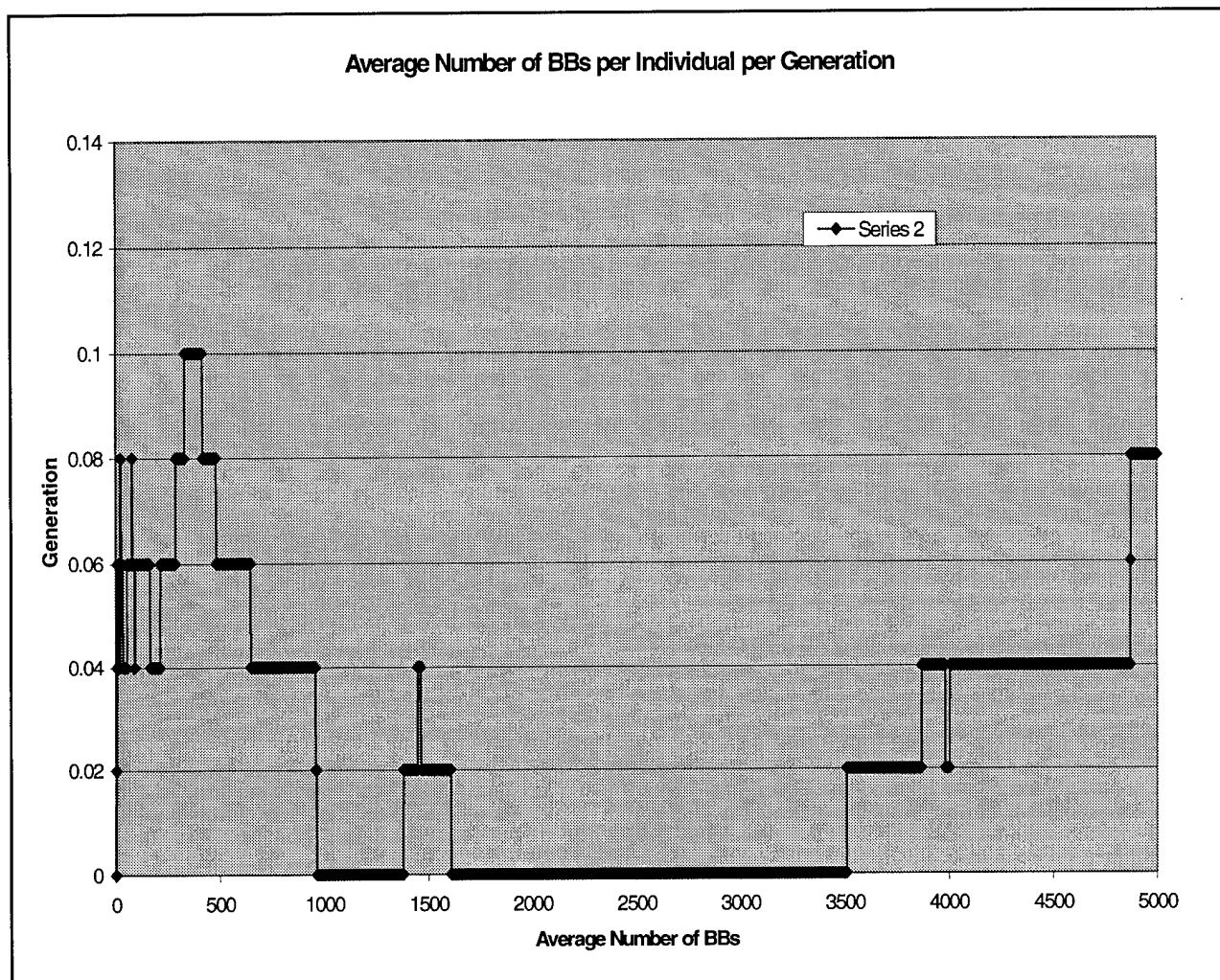


Figure 42: BBs Uncovered and Maintained for the pLLGA and Random Seed 2

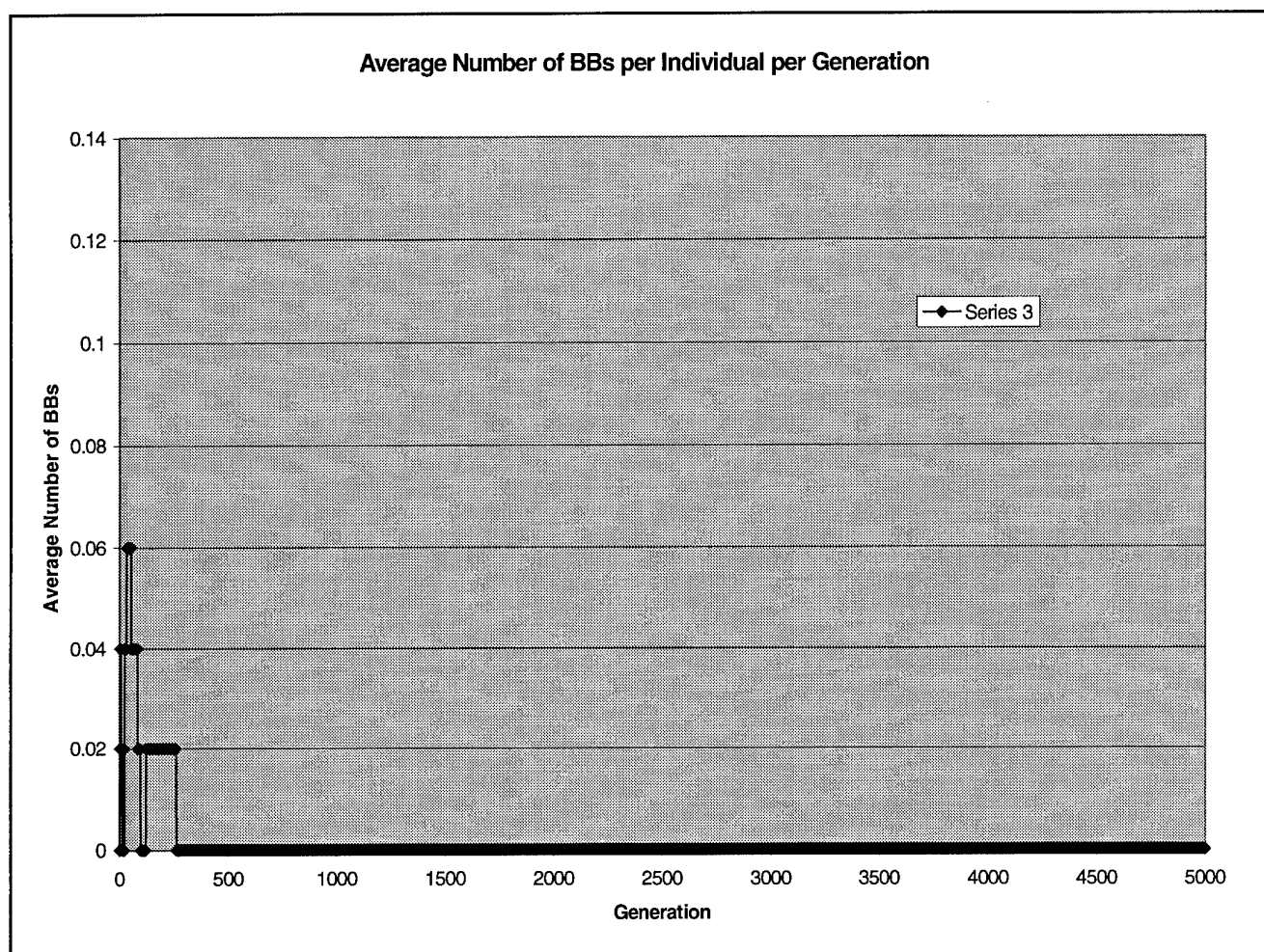


Figure 43: BBs Uncovered and Maintained for the pLLGA and Random Seed 3

Figure 41, Figure 42, and Figure 43 show this inability to maintain good BBs when the BBs are uncovered. Furthermore, **Figure 43** is the pLLGA test case that produced the most optimal overall energy out of all the test cases. As expected due to the ruggedness of the energy landscape, it is possible to calculate a rather low conformation energy and have nonrepresentational⁶⁷ dihedral angles of the protein. The ability of the cpLLGA to uncover and maintain good BBs is better, but this is only due to the constrained search space. It is not at all due to any algorithmic difference. **Figure 44, Figure 45, and Figure 46** represent the cpLLGA's ability to maintain BBs using the same random seed as in **Figure 41, Figure 42, and Figure 43**, respectively.

⁶⁷ The dihedral angles do not represent the dihedral angles of the QUANTA™ "optimal" solution.

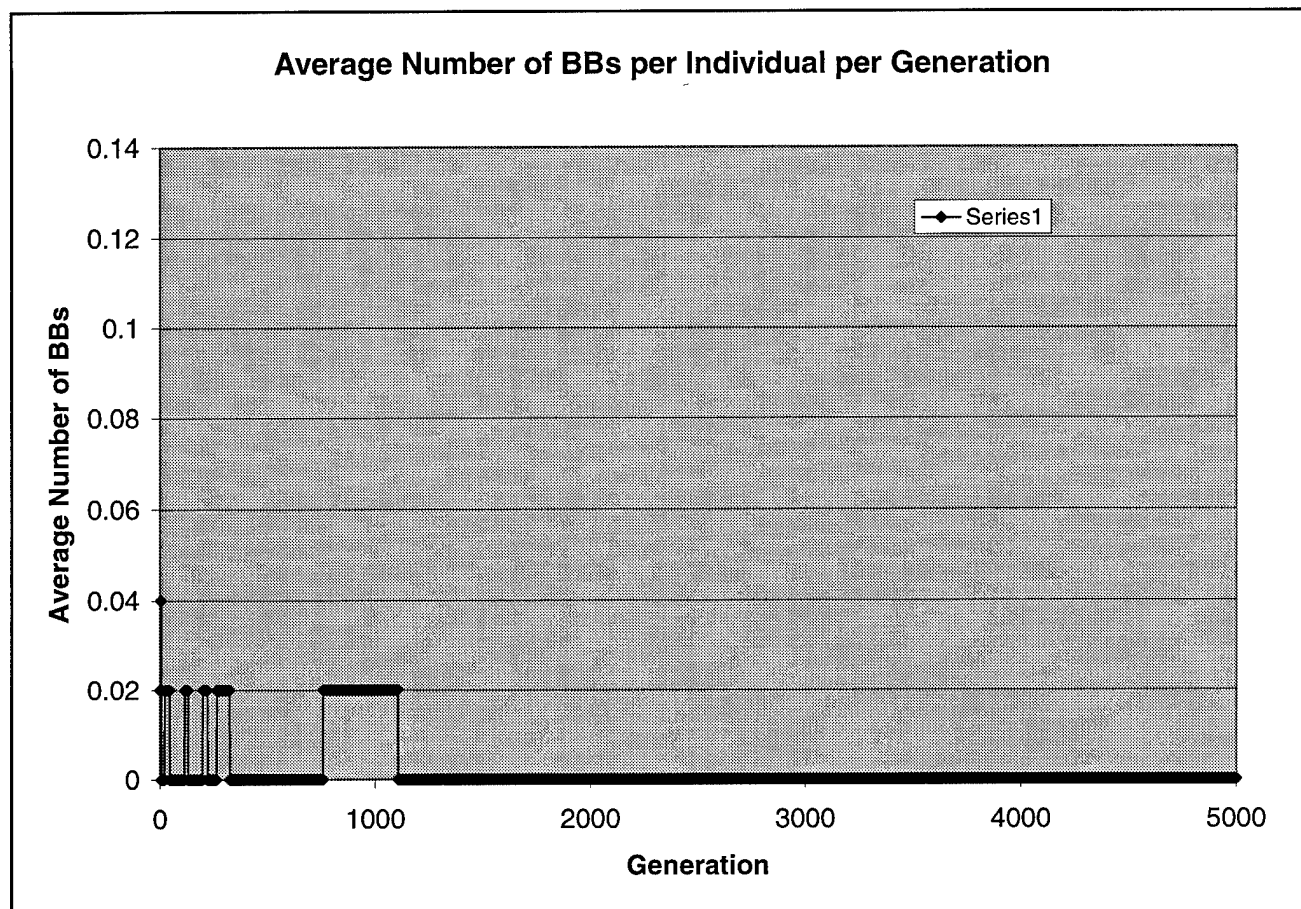


Figure 44: BBs Uncovered and Maintained for the cpLLGA and Random Seed 1

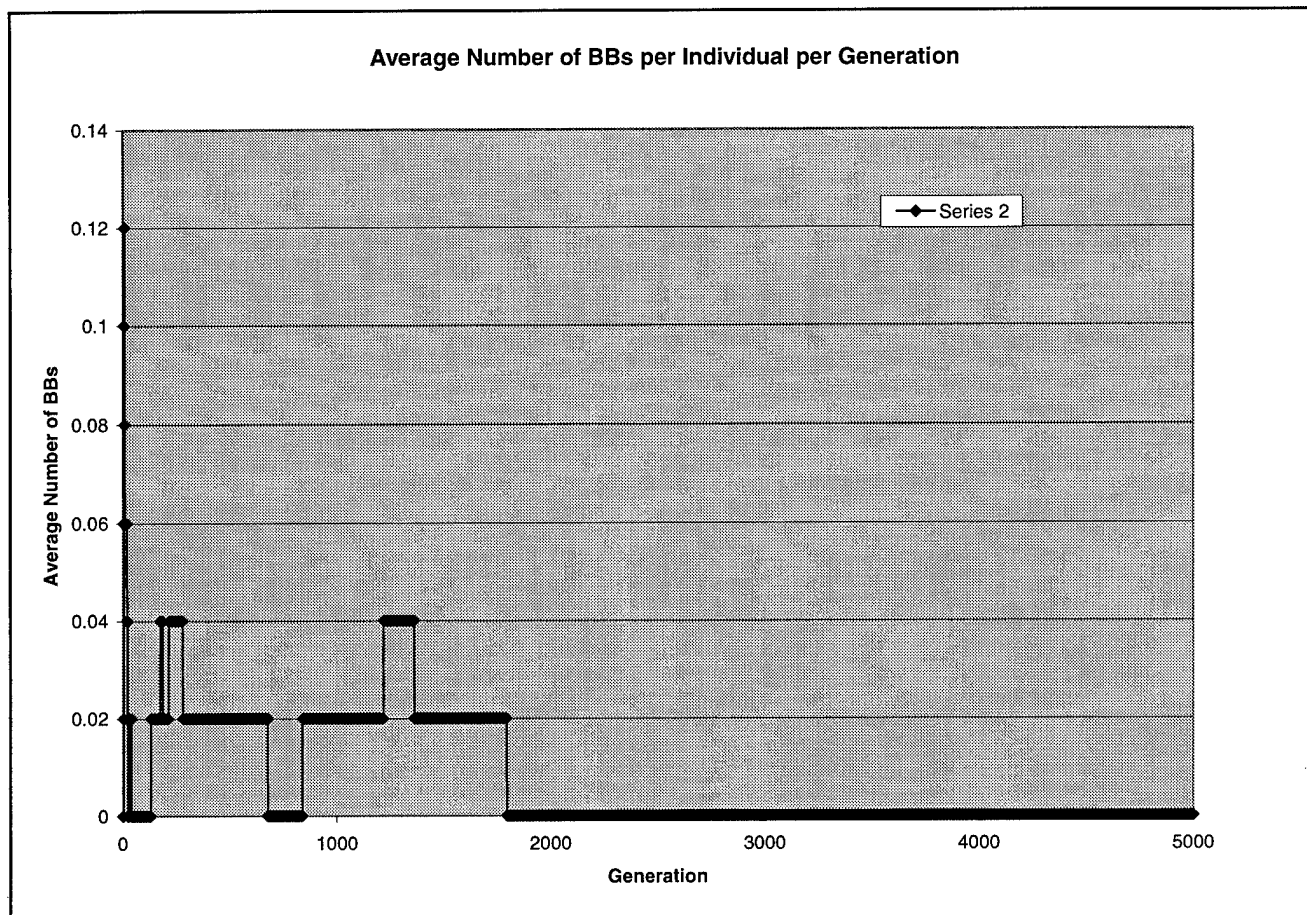


Figure 45: BBs Uncovered and Maintained for the cpLLGA and Random Seed 2

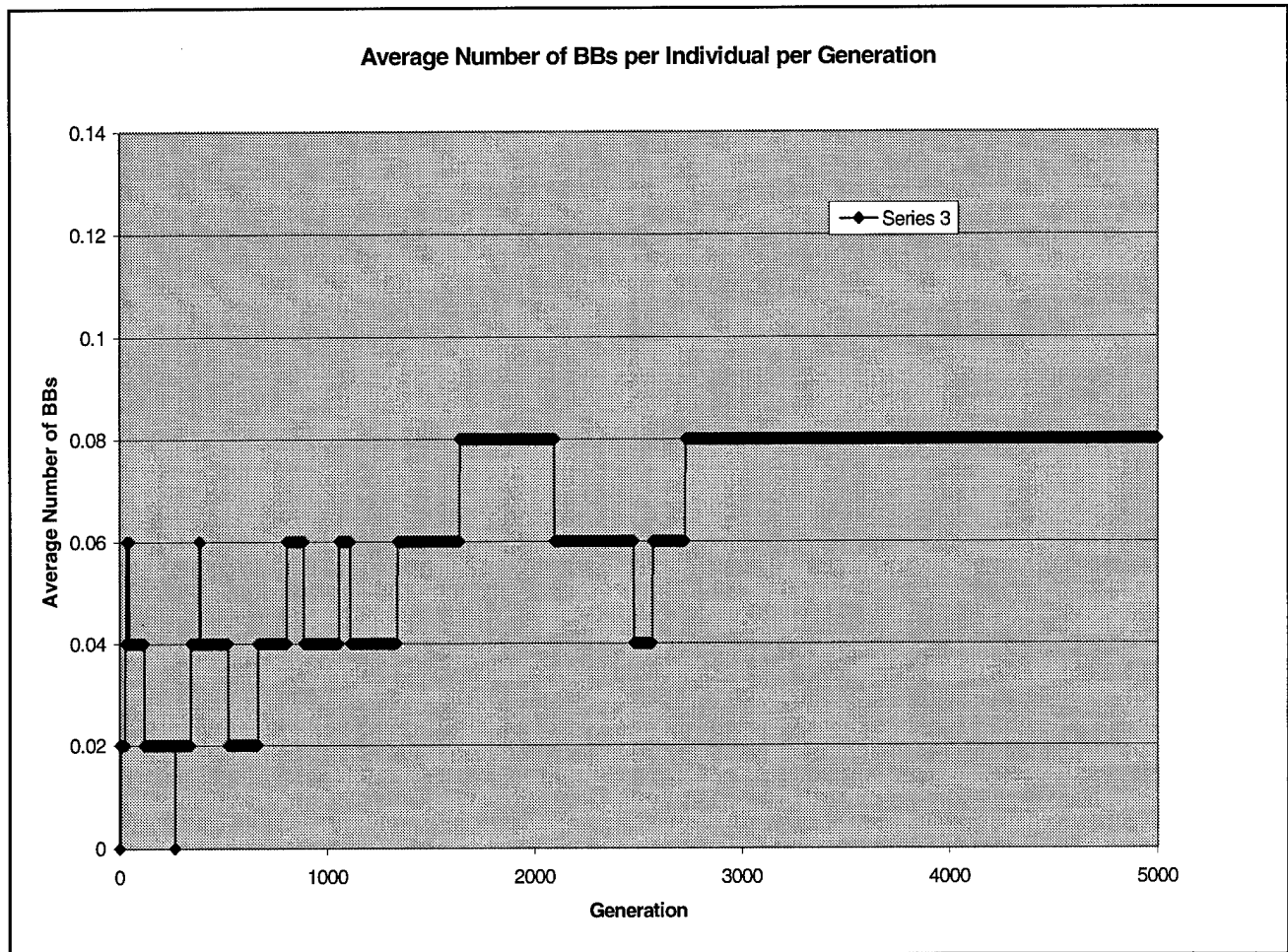
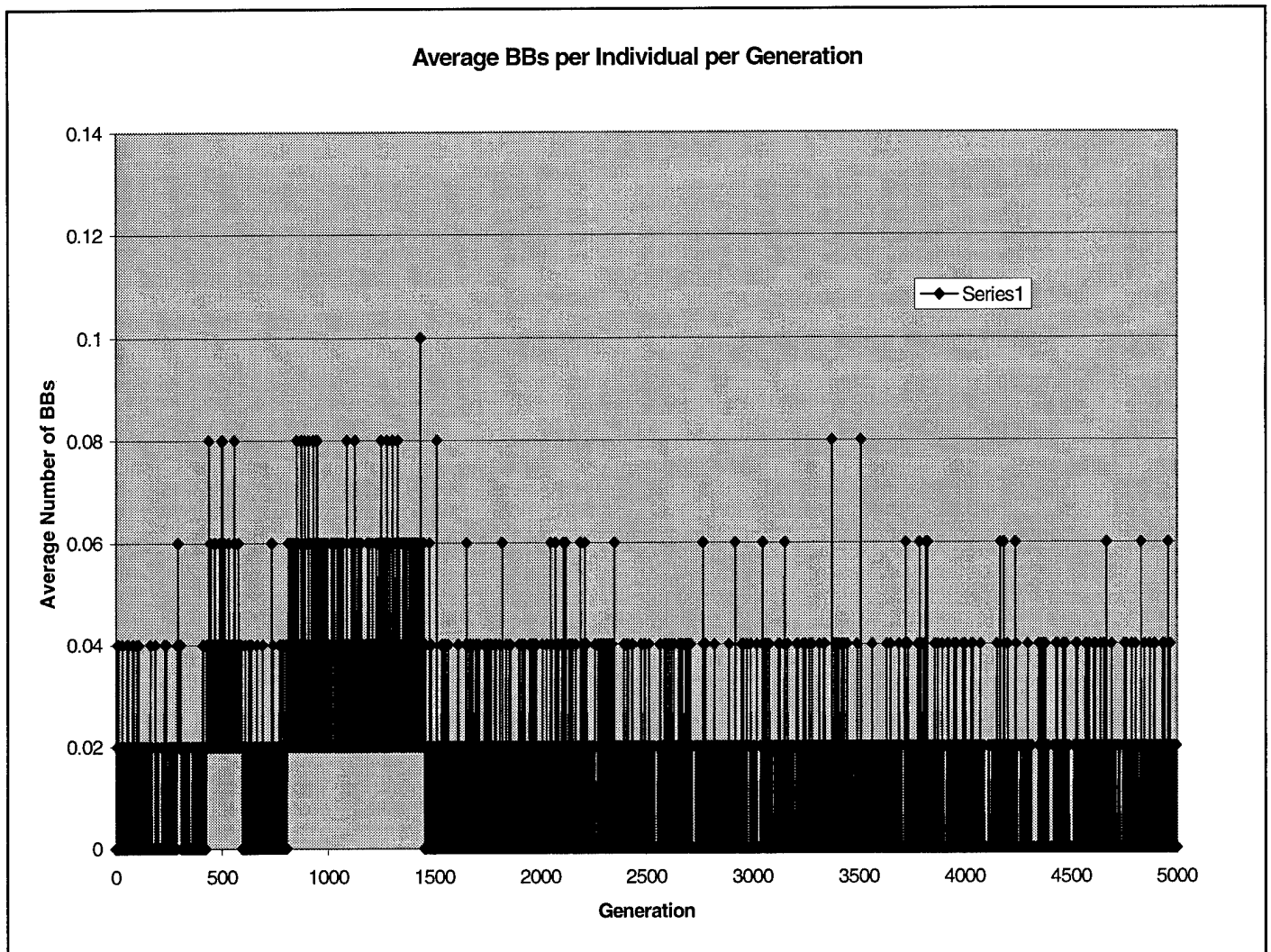


Figure 46: BBs Uncovered and Maintained for the cpLLGA and Random Seed 3

Secondly, it was decided to increase the selective pressure of the LLGA by changing the selection operator to see if this would increase the LLGA's ability to maintain good BBs. Therefore, the selection operator was changed to keep the best parent and the best child from the group of two parents and their two offspring. As can be derived from **Figure 47** and **Figure 48**, using this selection operator made the cpLLGA thrash in respects to its ability to finding and maintaining good BBs and converging to a particular location in the search landscape. **Figure 48** shows the cpLLGA's inability to converge after 5,000 generations (250,051 evaluations). Due to time constraints, we were not able to execute this test past the 5,000 generation mark. We presume that the most optimal energy uncovered will not change appreciable because according to the data the minimum energy had not changed since the 2,500th generation.



**Figure 47: BBs Uncovered and Maintained for the cpLLGA, Random Seed 3, and
the New Selection Operator**

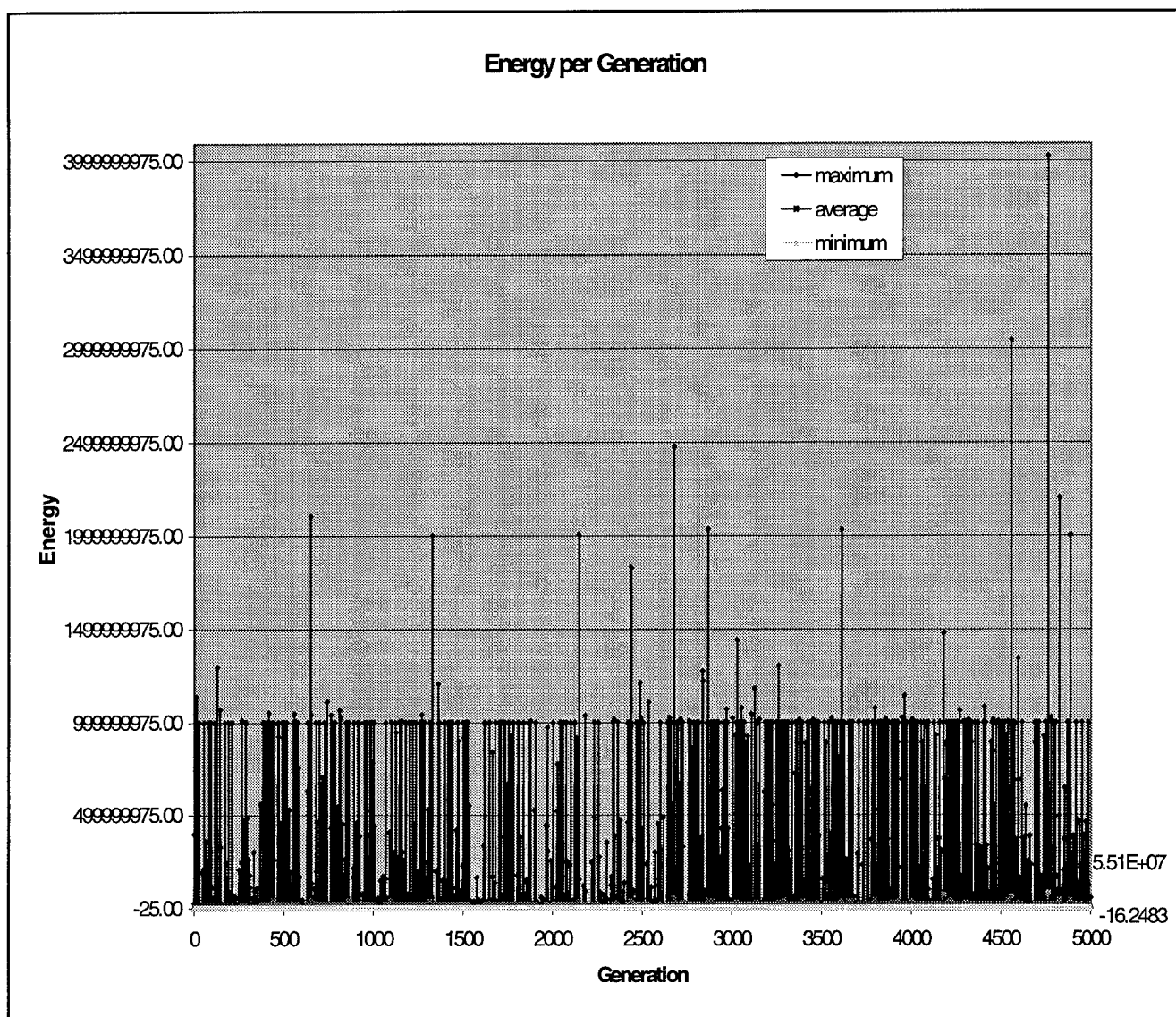


Figure 48: Energy Characteristics of the New Selection Operator

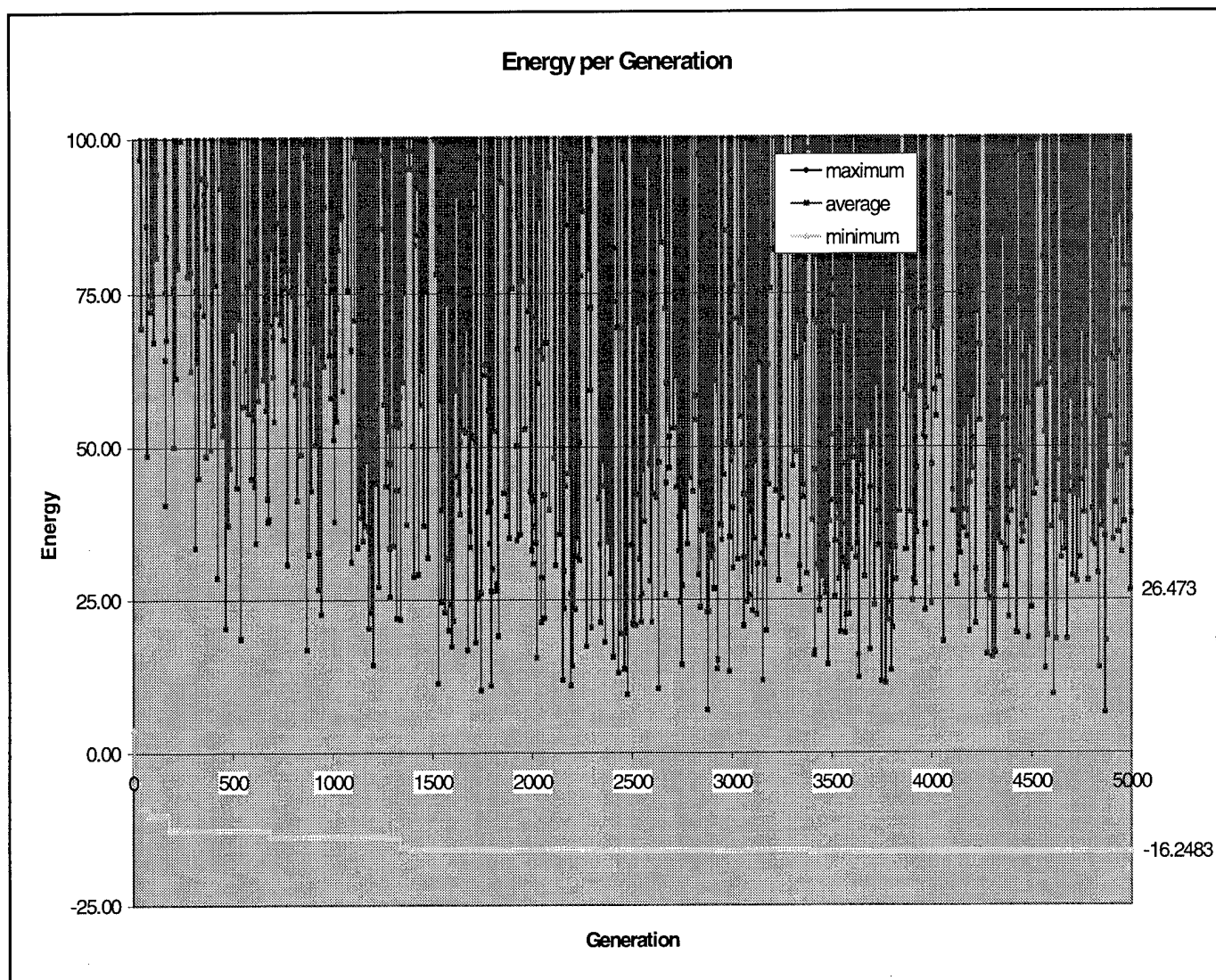


Figure 49: Energy Characteristics of the New Selection Operator (smaller scale)

6.7 Energy Landscape Visualization

Finally, we were able to incorporate 11,189 [Met]-Enkephalin configurations uncovered by the LLGA into a visualization of the landscape. **Figure 50** shows these points. The first point on the graph in **Figure 50** is the first entry of **Table 31** and the last point is the second entry of **Table 31**. Therefore, **Figure 50** can be considered an energy landscape visualization of the points between

```
0000000000101010011001001010001011100010001110000110001100011111011101
1101001001000101001111100010110110001101001101011110111100110010101100
```

101110100010000001101101010001000011011010011111111011010101111000110
101001101000011110101000010000 and
11111111111111011000111010001110111111001000010000110000111101000001
1101010110001111011101101110001011000100000110001011000011011101000101
0110110000100010111110001111001100010100001101101111011010110011011100
11110101110000010110010101100. **Table 32** and **Table 33** indicate the dihedral
angles represented by these two molecules of [Met]-Enkephalin.

Binary Chromosome Representation	Energy
00000000001010100110010010100010111000100011100001100011 00011111011101110100100100010100111110001011011000110100 110101110111001100101011001011101000100000011011010100 0100001101101001111111101101010111100011010100110100001 1110101000010000	919437.230312
1111111111111101100011101000111011111110010000100001100 00111101000001110101011000111101110110111000101100010000 01100010110000110111010001010110110000100010111110001111 00110001010000110110111101101011001101110011110101110000 0101100101011001	72282423.106913

Table 31: Limits of the Landscape Visualization

Residue	Dihedral Angle (degrees)						
	Φ	Ψ	ω	χ_1	χ_2	χ_3	χ_4
Tyr	-180.000	58.360	-75.938	-119.531	81.562	54.140	—
Gly	79.453	-100.898	17.227	—	—	—	—
Gly	167.695	115.664	-150.82	—	—	—	—
Phe	138.867	-40.430	-104.414	2.109	118.477	—	—
Met	85.430	159.609	5.62	-161.016	44.648	127.266	-137.109

Table 32: Dihedral Angles for Molecule One of the Visualization

Residue	Dihedral Angle (degrees)						
	Φ	Ψ	ω	χ_1	χ_2	χ_3	χ_4
Tyr	179.648	172.969	-16.523	114.258	-28.125	165.586	—
Gly	134.649	-133.594	-111.445	—	—	—	—
Gly	112.852	120.234	-93.164	—	—	—	—
Phe	-25.313	68.906	-145.547	16.523	21.094	—	—
Met	94.570	-102.656	-58.711	97.031	-103.008	127.617	-172.266

Table 33: Dihedral Angles for the Last Molecule of the Visualization

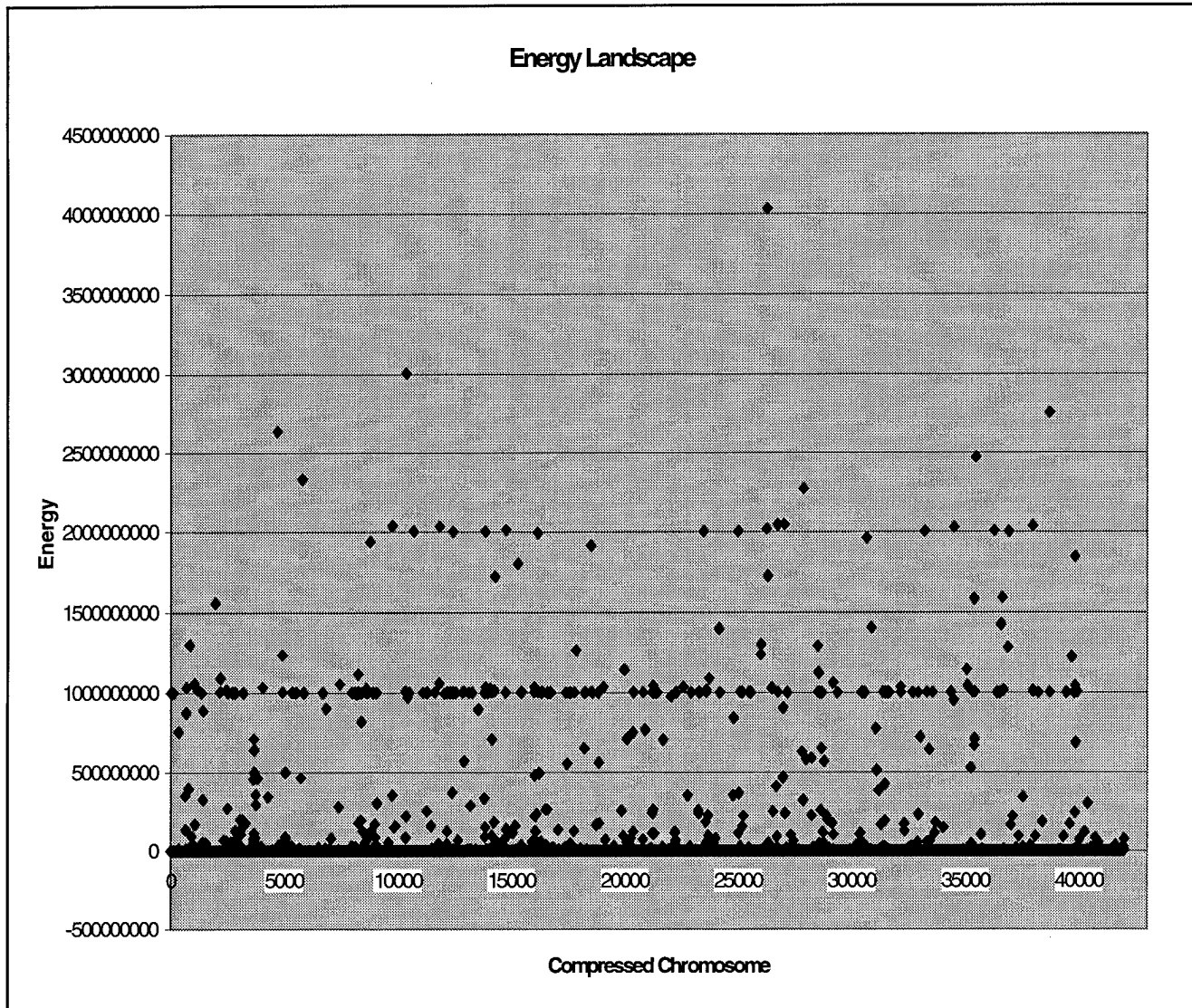


Figure 50: Energy Landscape Visualization

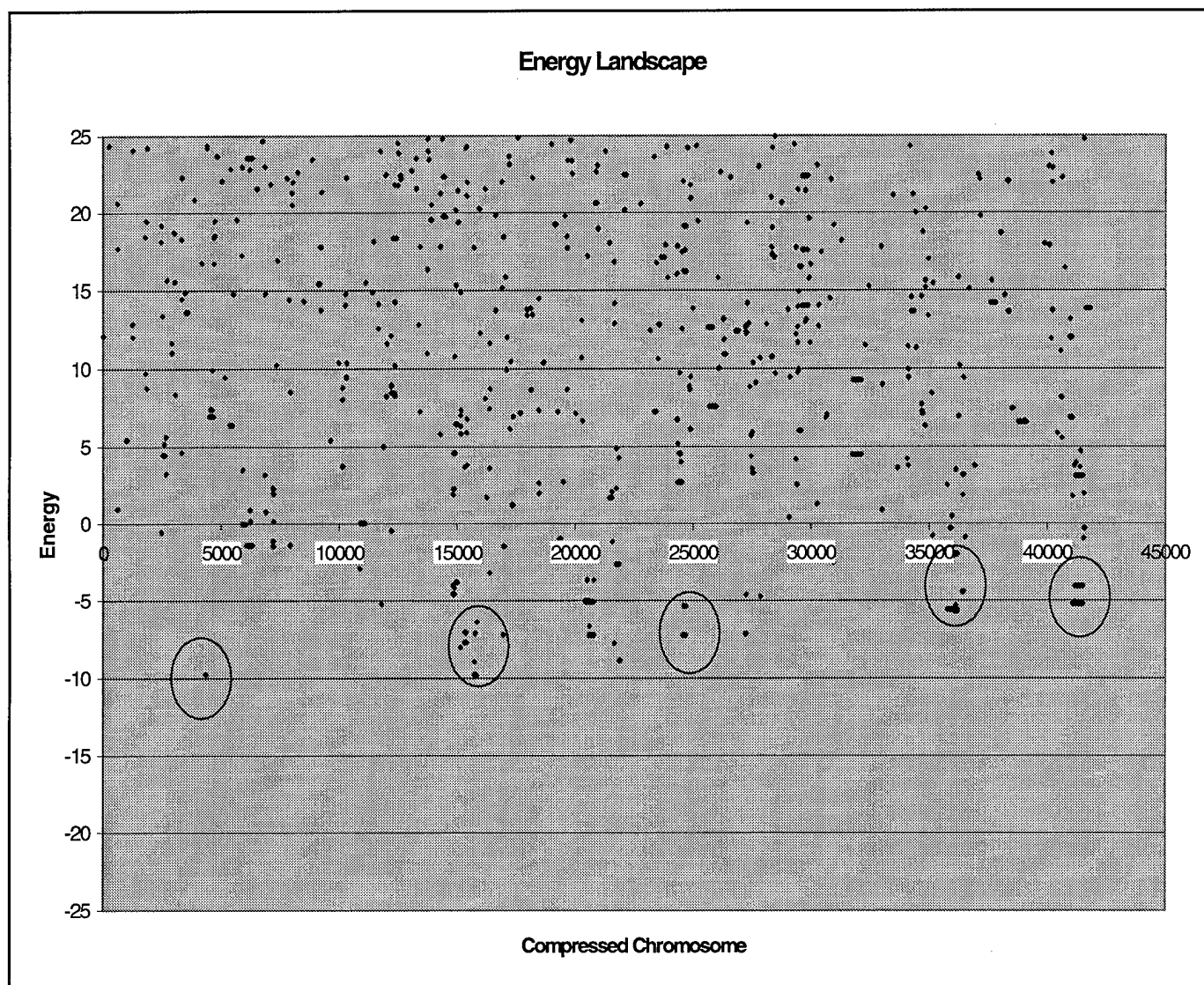


Figure 51: Condensed Energy Landscape Visualization

Figure 51 is a more limited view of the same energy landscape area bounded between +25 and -25 kcal/mole conformation energy. The visualization illustrates a picture of the landscape we did not expect to find. We had anticipated finding certain segments of the landscape that smoothly dipped into lower energy regions. But **Figure 51** clearly indicates that many of the low energy conformations are at the bottom of steep troughs in the landscape. Furthermore, we must consider that **Figure 51** is a 2-dimensional representation of the true 25-dimensional landscape. Therefore, the widths of these energy troughs depend on the sensitivity of each independent variable. For

instance, if we view the energy trough by pairwise independent variables⁶⁸ in order to determine the dependency the relationship represents, we may find that some of the relationships indicate wide valleys while other relationships between the independent variables are narrow chasms. The narrow chasms are of great concern to us because they represent high sensitivity. Any change in the two variables representing the chasm would trigger an enormous change in the protein's calculated energy. The first and third circled areas of **Figure 51** represent possible deep chasms, whereas the second, fourth, and fifth circled areas seem to indicate possible wide valleys.

These two figures do represent a "general idea" we have maintained for years. Basically, that the energy landscape of [Met]-Enkephalin is very rugged. Our visualization emphasizes that the landscape is extremely rugged⁶⁹ and it suggests the possibility that the landscape is also irregular. Furthermore, these figures indicate why a deterministic search method would flounder. Because the landscape is obviously riddled with low energy values surrounded by steep barriers, a deterministic method would enter the first low laying region⁷⁰ of the search space and not be able to escape. On the other hand, simulated annealing, which in some respects is similar to a deterministic search, is able to initially escape the first few local minima encountered because the progressively stronger penalizing function has not become sufficiently strong enough to anchor the algorithm to any particular local. As the penalty function increases, there would be more of a tendency to become trapped at the next best local minima.

6.8 Summary

Chapter 1 presents the objectives for this thesis. This chapter presents empirical results from the experiments designed in **Chapter 5** to meet those objectives. The performance of the LLGA, pLLGA, cLLGa, and the cpLLGA are compared using several different efficiency metrics. It is recommended that further test be conducted to statistically characterize this results⁷¹. Finally, a portion of the search landscape was revealed through our proposed visualization methodology.

⁶⁸ In this representation the two independent variables would be on the x and y-axis and the energy of the z-axis.

⁶⁹ The landscape turned out to be much more rugged then what we anticipated.

⁷⁰ I.e. find the first local minima.

⁷¹ **Appendix J** discusses possibly methods.

7.0 Conclusion and Recommendations

This investigation integrating the Protein Structure Prediction (PSP) problem and the Linkage Investigating Genetic Algorithms (LIGAs) over the past 18 months involved literature reviews, re-engineering the LLGA design and source code, uncovering esoteric aspects of AFIT's CHARMM energy model implementation, designing and implementing a visualization methodology, and designing and executing the appropriate experiments. These efforts have culminated in the realization of the goals we set forth in **Chapter 1**. The following two sections look at what we were able to conclude, the contributions from this research and some recommendations for future research.

7.1 Conclusions

Our application of the Linkage Learning GA (LLGA) to the PSP problem resulted in the conclusion that the LLGA is an inefficient software application shown in **Section 6.1**. Our analysis clearly indicates that the LLGA does not parallelize as well as past AFIT GA implementations. This is primarily due to the granularity of the parallelizable portions of the LLGA algorithm. Furthermore, the effectiveness of the LLGA is worse than previous implementations. The effectiveness of the algorithm may be increased by investigating methods to incorporate building block information. We understand from our re-engineering of the LLGA source code that the LLGA does not explicitly use the information gained by comparing the chromosomes of the population to the building block template. Therefore, the LLGA is basically accomplishing additional comparisons/calculations that are unnecessary and lead to additional computational overhead without benefiting the search process.

The algorithmic contribution of the LLGA is its ability to overcome the disruptive effects of crossover. LLGA accomplishes this task by the inclusion of introns into the chromosomal representation. But, again, our research has shown that as we moved from small contrived academic problems to complex real-world applications the number of introns required by the LLGA explodes! This indicates the LLGA has an inability to scale to larger problems, which severely hampers its usability in real-world applications. As the number of introns grows the LLGA's search for the canonical form takes an increasingly larger toll on the computational performance of the algorithm. Again, this puts a damper on our desire to recommend the use of the LLGA in large real-world

applications⁷². Furthermore, in our experiments we used 19 times more introns than exons and we were still unable to maintain good building blocks (i.e., cancel the disruptive effect of crossover). This is discussed in **Section 5.3** and **Section 6.6**.

On the other hand, our attempt to include problem domain constraints directly into the decoding of the chromosome shows great promise. The additional computation overhead of this scheme is negligible! Furthermore, this transformation process does not lead to “islands of feasibility” where the GA becomes trapped as it did for Kaiser [37]. See **Section 6.3** and **Section 6.4**. Our testing has shown the cpLLGA is better at uncovering and propagating building blocks. Shown in **Section 6.6**.

Finally, our novel approach to visualizing the PSP landscape traversed by any GA shows an ability to gain insight into both the algorithmic processing and the possible energy landscape structure of the specific protein in question. Insights gained so far from our limited instantiated visualization has substantiated our current notation that the PSP energy landscape has an extremely rugged and irregular domain. The visualization of the landscape is shown in **Section 6.7**.

7.2 Contributions

The general conclusions drawn from this research lead to the following contributions to the algorithm domain and the PSP problem domain. The key products produced as part of this thesis effort are:

- 1) An ineffective and inefficient building block propagating GA when applied towards the PSP problem.
- 2) An insightful solution space visualization methodology.
- 3) An effective and portable PSP search space-bounding function that can be incorporated into any GA.

7.3 Recommendations

Our recommendations for future research efforts lie in modifying the LLGA, incorporating more problem domain information into the process in order to further constrict the search space, and redirection of our primary focus.

Modifying the LLGA algorithm is a possibility for future research. The LLGA chromosomal data structure could be re-engineered to make use of the information

⁷² For our 240 bit chromosome the LLGA required 4650 introns for a crossover disruptive probability of 0.05%.

gleaned from the building block template. A possible way to do this is by adding a weight to the chromosome that accounts for the inclusion of a building block within the canonical form. This would increase the LLGA's ability to recognize building blocks within chromosomes of the population and maintain this information. This concept is similar to the infected genes evolutionary algorithm [86].

For the PSP problem domain, we recommend further research into the applicability of constraints beyond Kaiser's work [37]. The search space for the PSP problem constrained solely by Ramachandran constraints is still enormous! But there are other avenues of constraints, which could be included into our model such as affinity to form hydrophobic or hydrophilic structures, side-chain placement strategies, and steric constraints. These "other" constraints could lead to an even smaller, yet still intractable, search space.

Furthermore, AFIT's model needs to be validated against larger molecules. Current and past research has focused on small proteins⁷³, but AFIT has yet to show that the combination of any genetic algorithm and the AFIT's CHARMM energy model can handle larger proteins consisting of hundreds of residues. This research should provide additional insight into the general applicability of GAs to the PSP problem.

Finally, there are often conflicting methods for calculating a protein's tertiary structure. These different methods could be employed in a multi-objective approach to afford the biochemist greater insight into the PSP problem. **Table 34** indicates possible secondary fitness functions:

Category	Characteristic
Electromagnetic	Energy transfer or reflection
Entropy	Information content and (dis)order
Environmental	Environmental benefit or damage
Geometrical	Structural relationships
Physical (Energy)	Energy emission or transfer
Physical (Force)	Exerted force or pressure

Table 34: Possible Fitness Functions

⁷³ [Met]-enkephalin is a pentane.

Appendix A. Background on the Protein Folding and Protein Structure Prediction Problems

This appendix contains background material on the protein folding and protein (or polypeptide) structure prediction problems, most of which has been presented in previous AFIT theses, particular those of Brinkman (18) and Gates (15). Section A.1 defines terminology in the biochemistry domain. Section A.2 describes the expensive experimental techniques used to determine the structure of proteins. Finally, Section A.3 examines various models used to predict the structures of polypeptides and proteins.

The protein folding problem (PFP) has been recognized as a National Grand Challenge problem in biochemistry and high-performance computing (11). The challenge is to find a method to predict the three- dimensional geometry of a protein based on the sequence of its components. A solution, which would provide knowledge about the function(s) of individual proteins, is also the first step toward solving the inverse protein folding problem (IPFP) (8, 71). The goal of the inverse folding problem is to determine a sequence (possibly more than one) that folds to a specified three- dimensional structure.

The difference between the two problems is best characterized by the capability that their solution would provide. A PFP solution would enable the evaluation of many proteins in a search for one with a specific property or function. In contrast, an IPFP solution would provide a direct mechanism to design a protein with specified characteristics (8:25--26). Possible applications include: pharmaceuticals with few or no side effects; energy conversion and storage capabilities (similar to photosynthesis); biological and chemical catalysts and regulators; angstrom scale information storage; and possible optical/chemical shielding from harmful radiation sources (8:25) (71:5) (93).

A.1 Introduction to Proteins and Associated Terminology

Proteins (polypeptides) are linear sequences of the 20 naturally occurring amino acids. Each amino acid consists primarily of three common backbone atoms (a nitrogen and two carbons $[N-C_{\alpha}-C_{\gamma}]$ bonds, called the side-chain (S_i), connected to the C_{α} carbon atom. A particular protein is defined by its unique amino acid sequence, which is known as the primary structure of the protein (8:24)(71:2)(69:49).

As the amino acids form into proteins via peptide bonds, they give up a water molecule. The linked amino acids are called residues. Figure 32 depicts a generic protein composed of three residues (amino acids). In most contexts, the terms amino acid and residue are used interchangeably. The primary structures of approximately 50,000 naturally occurring proteins are currently known and this number is expected to double every year, due largely to the Human Genome Project and the ease with which sequences are experimentally determined (71:5)(91). In fact, the sequence determination and also fabrication is fully automated.

Subsequences of proteins tend to exhibit regular patterns. Two common patterns are α -helices and β -sheets. These describe the secondary structure of a protein (8:24). Secondary structures result only when at least four or five consecutive amino acid residues have similar ϕ and ψ values (57). Some researchers are investigating the utility of predicting secondary structure as the first step of tertiary structure prediction (69:50). This technique has had limited success. The problem is that even though certain residues are found more frequently in a specific secondary structure, the greatest preference is only twice that of other secondary structures. In most cases, the preference is much smaller (108:422). Table 22 identifies the values for (ϕ , ψ) angle pairs that according to Horton (57) ideally define commonly occurring secondary structures.

Secondary Structure	Phi (ϕ)	Psi (ψ)
α -helix (right hand)	-57	-47
α -helix (left hand)	57	47
3_{10} Helix (right hand)	-49	-26
Antiparallel β -sheets	-139	235
Parallel β -sheets	-119	113
Collagen Helix	-51	153
Type II turn (second residue)	-60	120
Type II turn (third residue)	90	0
Fully extended chain	-180	-180

Table 35: Phi & Psi Pairs of Common Secondary Structures

The three-dimensional structure of a protein is the major determinant of its function. This three-dimensional shape is called the *tertiary structure* or *conformation* of the protein. Proteins assume their *native* conformation, which is unique and typically compact, in their natural biological environment (typically in aqueous solution, at neutral pH and

20—40° C) (8, 71). A protein in its native conformation is only slightly more stable than the various conformations with marginally higher energies. Normally, there is only a 10 kcal/mol energy difference between the completely folded and unfolded conformations. This single fact is responsible for the major difficulty of the protein folding problem (8:24-25) (71:2--4) (69:50).

There are two principle coordinate systems frequently used to identify the position of the atoms in a molecule. The Cartesian coordinate system uses a three dimensional coordinate $(x_i; y_i; z_i)$, $1 \leq i \leq n$, where n is the number of atoms in the molecule. An arbitrary atom, usually $C_{\alpha 1}$ is assigned to the origin. This system is most useful to compute the distance, $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ between two atoms. With this system each molecule has $3n$ degrees of freedom.

Internal coordinates is the other coordinate system. The dihedral angle approach defines position of all atoms in a protein from the position of one atom (usually at the origin), the *bond length* of each covalently bonded pair of atoms, the *bond angle* formed by each triplet of bonded atoms, and the *dihedral angle* formed by each bonded group of four atoms (see **Figure 56**, **Figure 57**, **Figure 58**). Given this set of parameters, every protein has $3n-6$ degrees of freedom where n is the number of atoms. However, the bonds and bond angles are relatively rigid, therefore the independent dihedral angles are left as the only dominant factor to determine the tertiary structure of a protein. Hence, the degrees of freedom are effectively reduced by a factor of approximately $2/3$ (8:26) (69:50).

Each amino acid contains a ϕ , ψ , and ω dihedral angles and zero or more χ_i dihedral angles as shown in **Figure 1**.

If we discretize the domain of the dihedral angles so that there are d possible values, then the size of the search space is given by d^N where N is the number of independently variable dihedral angles. Given a very coarse 20° discretization of the $0 - 360^\circ$ and a small protein with 24 independently variable dihedral angles, the search space contains $18^{24} \approx 1.3 \times 10^{30}$ conformations. **Table 23** shows the time required to enumerate the search space on current and envisioned high performance computers (under the optimistic assumption of one evaluation per clock cycle) (107:7)! (Giga-, Tera-, and Peta-FLOP computers perform 10^9 ; 10^{12} , and 10^{15} floating point operations per second, respectively) Therefore, if we hope to find the single native

conformation of a protein, we must have access to efficient search algorithms that severely prune the search space.

Computer Speed	Execution Time (years)
1 GigaFLOP	≈ 41 trillion
1 TeraFLOP	≈ 41 billion
1 PetaFLOP	≈ 41 million

Table 36: Enumeration Time of 1.3×10^{30} Search Space at One Solution per Clock Cycle

Appendix B. Current Methods for Protein Structure Prediction

B.1 Introduction

A protein consists of a sequence of amino acids. Each acid is identified by an attached sidechain [18]. A single sidechain group is a rigidly connected sequence of atoms commonly referred to as a “peptide unit” or a “residue” [19]. Each residue is read left-to-right beginning with the amino and ending at the carboxyl terminal [19] (see **Figure 1**). The sequence of amino acids, joined together by peptide bonds (i.e. several peptide unit or residues represented as a 1-dimensional model), form the basis for what is referred to as the primary structure of a protein. These structures help us to understand the chemical configuration of the protein, but the biological role of a particular protein is defined by its tertiary structure [20]. The atomic forces interacting between the atoms within the protein molecule form the tertiary structure. The tertiary structure is a twisted, grooved, helixed, sheeted, and creviced 3- dimensional structure. It is these crevices and grooves of a protein’s complex folds that allow the protein to attach to other molecular structures and define its function [20]. Within these twisted and tangled structures are regularly occurring patterns called secondary structures. Secondary structures are believed to be the stepping stones in the process of folding a protein [22]. The secondary structures are classified either as right- or left-handed alpha helices, beta-sheets, or random coils [21]. It is the final tertiary structure that is of utmost importance to biochemists and this state is called the natural molecular conformation. The protein folding problem can, therefore, be described as searching for this natural conformation state given only the primary structure of a protein. Knowing the structure of biological molecules allows scientists to better understand how they work and can lead to better drugs and treatments for disease.

B.2 Practical Methods for Calculating a Protein’s Native Structure

In order to understand and manipulate proteins, we must be able to reliably predict the tertiary structure of the protein in a reasonable amount of time. Generally, there are three different methods to determine the conformation state of a protein: X-ray Crystallography, Nuclear Magnetic Resonance, and Computational Models. X-ray crystallography and nuclear magnetic resonance spectroscopy are direct methods of

measuring the position of each atom within a protein. These methods are extremely time consuming and laborious! Computational modeling, on the other hand, is somewhat less time consuming and easier to conduct, but these methods are approximations and may not precisely reflect the native structure of a particular protein. Although, computational modeling has many shortcomings, it is still the greatest area of interest to biochemists because this form of calculating the native structure provides the greatest possibility of shortening the gap between the discovery of a new protein and learning its conformational structure. This section provides a brief overview of the x-ray crystallography and nuclear magnetic resonance spectroscopy, and an in-depth look at several different forms of computational modeling.

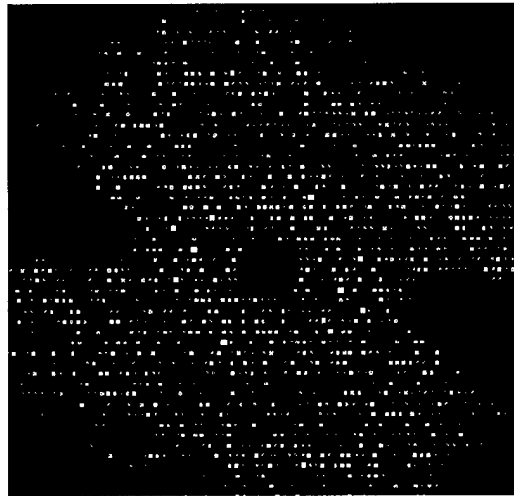
B.2.1 X-ray Crystallography

Scientists have used X-ray diffraction patterns since the early part of this century to aid their studies of molecules. X-ray crystallography was the first technique to reveal the precise 3-dimensional position of most of the atoms in a protein. In order to understand how x-ray crystallography works, we must remember that an atom consists of a nucleus surrounded by electrons. The electrons scatter the x-rays in all directions. The intensity of scattering from a given atom is dependent largely on the number of electrons present, and can be thought of as a fingerprint for a particular element by the "atomic scattering factor" [27]. If a periodic array of atoms is present, constructive and destructive interference patterns result. This observed diffraction only is seen in certain directions and for a given orientation of the periodic array with respect to the x-ray source. Since crystals consist of molecules arranged periodically, a crystal acts as a nearly perfect diffraction grating for the x-rays [27]. In order to "see" an object, its size has to be at least half the wavelength of the electromagnetic radiation being used to view it [28]. Therefore, the x-rays routinely used in crystallography have wavelengths of 0.7 to 1.7 Angstroms [27].

The simplified steps to this procedure are as follows [22]:

- 1) First, crystals of the protein of interest are needed. NOTE: The quality of the crystal determines the ultimate resolution of this analysis.
- 2) The protein crystal is mounted in a capillary and positioned in a precise orientation with respect to the x-ray beam and film. Precise motion of the crystal results in an x-ray photograph consisting of a regular array of spots. (See **Figure 52.**)

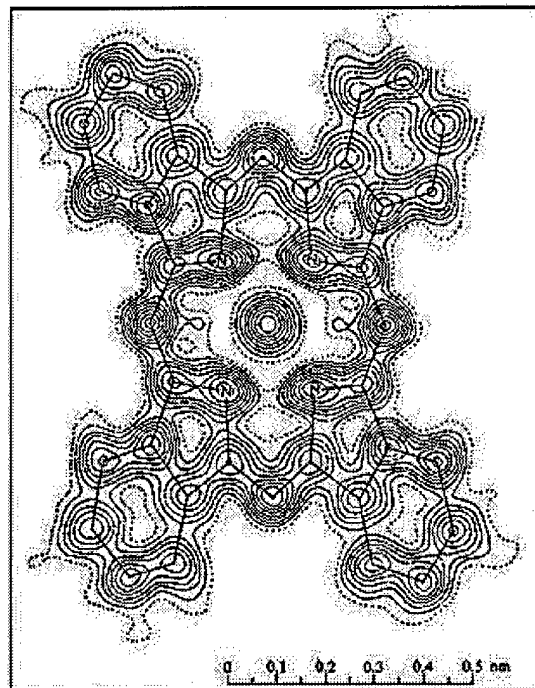
Figure 52: X-ray Diffraction Pattern for protein Lac Repressor [27]



- 3) The intensities of the spots are measured. These intensities are the basic experimental data of the analysis.
- 4) Next, the image of the protein is reconstructed by applying a Fourier transform, and an electron-density map is created. The electron-density map gives the density of electrons at a large number of regularly spaced points in the crystal. (See **Figure 53.**)

Figure 53: Electron Density Map [29]

- 5) Finally, the electron-density map is interpreted.



A resolution of 6 Angstroms (\AA) reveals the positions of the atoms in the backbone, but few other structural details. This is because atoms that comprise the polypeptide backbone are centered 5 and 10 \AA apart. Maps at higher resolutions are needed to delineate groups of atoms that lie from 2.8 to 4.0 \AA apart, and individual atoms that are between 1.0 and 1.5 \AA apart [22].

B.2.2 Nuclear Magnetic Resonance Spectroscopy

Nuclear Magnetic Resonance (NMR) spectroscopy was made possible by Felix Bloch and Edward Purcell in 1952. Until then, magnetic resonance was a measurable phenomena in which atoms were shot through a magnetic field as a beam [23]. I. I. Rabi laid this groundwork in the theoretical properties of NMR research, but it was Bloch and Purcell's development of NMR instruments that could measure this phenomena in bulk materials such as liquids and solids that open up the door for using NMR as a means to measure the natural state of proteins [23]. NMR spectroscopy is based on the measurement of the absorption of electromagnetic radiation in the radio frequency region between 4 and 750 MHz [24]. A simplified description of this technique follows:

- 1) The sample is submitted in a deuterated solvent and transferred into the NMR tube.
- 2) The tube is placed into a magnetic field.
- 3) A radio frequency pulse is then sent through the sample solution in order to orient the magnetic moments of the nuclei in the solution.
- 4) As the magnetic moments relax, they exhibit a free induction decay with time. The sample eventually relaxes to its equilibrium state.
- 5) The free induction decay is Fourier transformed into a NMR spectrum.

The relaxation process is highly informative about the macromolecular structure and dynamics because they are highly sensitive to both the geometry and motion [22]. The NMR spectrum displays the chemical shifts for the individual nuclei; and from these shifts, the structure of the compound can be determined [25]. A sample spectrum display is provided in **Figure 54**:

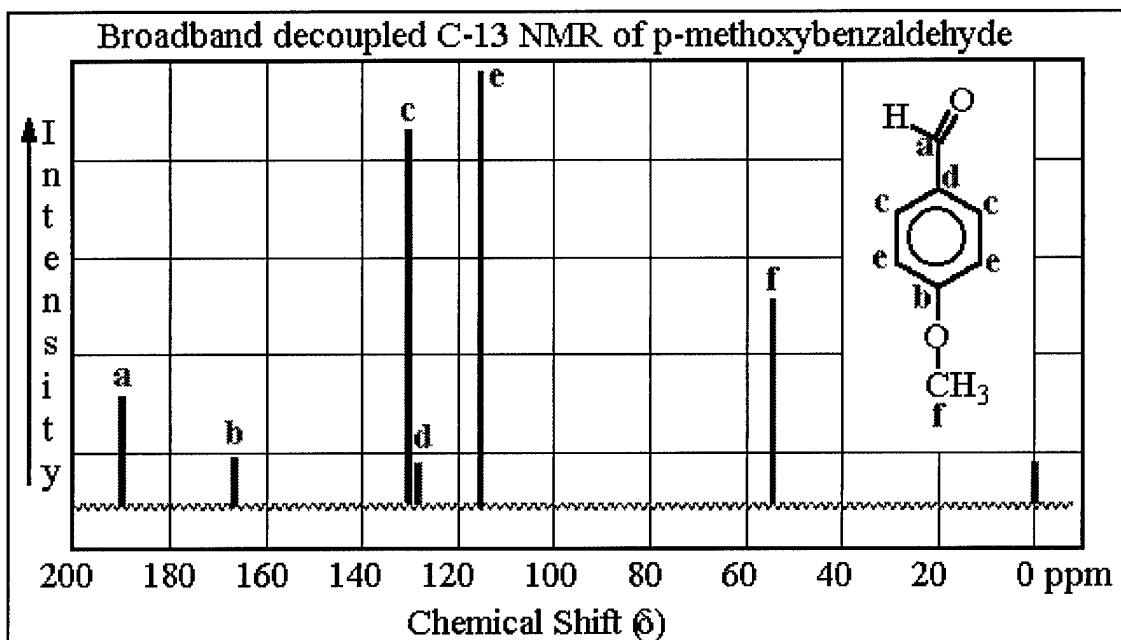


Figure 54: Sample 1D NMR Spectrum [26]

NMR spectroscopy is one of the most powerful tools available to chemists and biochemists for the elucidation of the structure of both organic and inorganic specimens.

B.2.3 Computational Models

Computational engines are used to calculate molecular energies and properties associated with these energies [30]. There are three major classes of computational engines: 1) Empirical, 2) Semi-Empirical, and 3) Ab Initio. Previous AFIT research on the PFP problem has centered on using semi-empirical engines. Therefore, the following discussions covering these three forms of computational engines will provide more depth and analysis into a particular semi-empirical method (i.e. the CHARMm energy model), but since one of the objectives of my thesis effort is attempting to use a secondary objective function when solving for the native state of the protein, additional information for a particular empirical method will also be presented (i.e. Schrodinger's Equation).

B.2.3.1 Empirical

Empirical methods use principles founded in molecular mechanics to describe molecular energetics in terms of a set of classical potentials. Molecular mechanics models are based on the following assumptions [31]:

- 1) Nuclei and electrons are lumped into atom-like particles considered rigidly spherical and having some net charge.
- 2) Interactions are based on springs (representing bonds between atoms) and classical potentials (representing forces between non-bonded atoms).
- 3) Interactions must be pre-assigned to specific sets of atoms.
- 4) Interactions determine the spatial distribution of the spherical atoms and their energies.

The objective of these molecular mechanics models is to predict the energy associated with a given conformation of a particular molecule. However, these energies have no meaning as absolute quantities (i.e., there is no “right” reference energy); only differences in energy between two or more conformations of a particular atom have meaning [31]. In other words, we cannot conduct a straight comparison between two conformations evaluated using different models nor can we adequately compare two different molecules using the same energy model.

B.2.3.1.1 Anatomy of a Molecular Mechanics Force-Field

A simplified molecular mechanics energy equation is:

$$\text{Energy} = \text{Stretching Energy} + \text{Bending Energy} + \text{Torsion Energy} + \text{Non-Bonded Interaction Energy}$$

Equation 24: Simplified Semi-Empirical Energy Equation

These potential functions and the data used for their evaluation are collectively called a “force-field” [30]. Separate potential functions are used to calculate *bond stretching*, *angle bending*, *bond twisting energies*, and *non-bonded interactions*. (See **Figure 55.**)

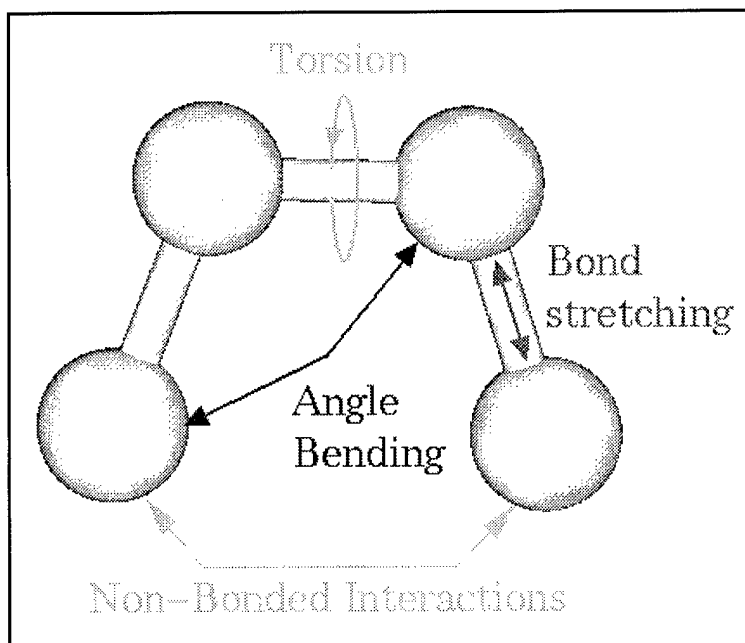


Figure 55: Overview of the Mechanical Molecular Model Forces

B.2.3.1.1.1 Bond Stretching Energy

The energy due to bond stretching (**Figure 56**) is based on Hooke's Law (see **Equation 25**).

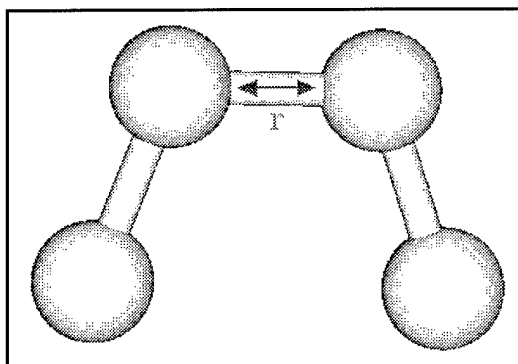


Figure 56: Bond Stretching

$$E = \sum_{\text{bonds}} k_b (r - r_o)^2$$

Equation 25: Bond Stretching Energy

where, K_b controls the stiffness of the bond spring, and r_o defines the equilibrium bond length [30].

Unique, K_b and r_o parameters are assigned to each type of bonded atom pair (e.g. C-C, C-H, O-H, etc.) [31]. The **Bond Stretching Energy** estimates the energy associated with the vibrations about the equilibrium bond length [31]. This model tends to break down as the bond is stretched to the point of dissociation.

B.2.3.1.1.2 Angle Bending Energy

Bending energy (E) is also based on Hooke's Law. (See **Equation 26.**)

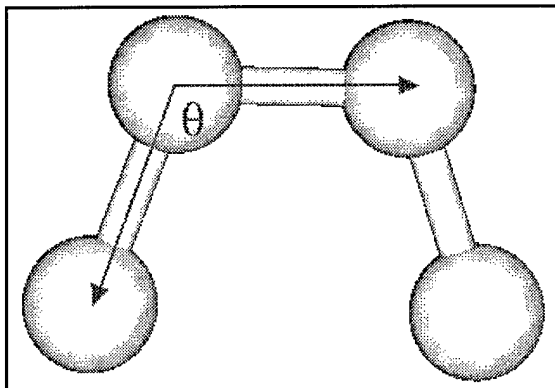


Figure 57: Angle Bending

$$E = \sum_{\text{angles}} k_{\theta} (\theta - \theta_0)^2$$

Equation 26: Angle Bending Energy

where, K controls the stiffness of the angle spring, and θ_0 defines the equilibrium angle [30].

The angle bending energy equation estimates the energy associated with the vibration about the equilibrium bond angle [31]. Unique parameters for angle bending are assigned to each type of bonded triplet of atoms (e.g. C-C-C, N-C-C, C-C-H, etc.) [31]. The larger the value of " k_{θ} ," the more energy is required to deform an angle from its equilibrium value by a given amount.

B.2.3.1.1.3 Non-Bonded Energy

Non-bonded energy represents the pair-wise sum of the energies of all possible interacting non-bonded atoms (i and j) within a molecule. This equation accounts for the van der Waals attractions and repulsions, as well as electrostatic interactions [31]. Van der Waals attractions occur at short ranges between atoms, and rapidly die off as the two atoms move apart by just a few angstroms [31]. Repulsion occurs when the distance between interacting atoms becomes slightly less than the sum of their contact radii [31]. Repulsion counteracts the effects of the van der Waals attraction, and is modeled by a function that is specifically designed to rapidly explode at close distances. The electrostatic interaction term serves to describe the smooth transition between these two regimes [31].

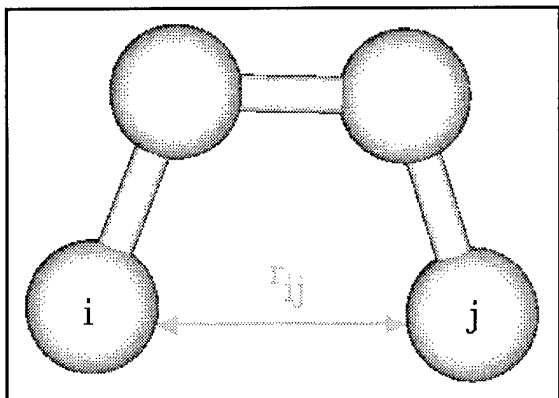


Figure 58: Non-bonded Interaction

$$E = \sum_i \sum_j \frac{-A_{ij}}{r_{ij}^6} + \frac{B_{ij}}{r_{ij}^{12}} + \sum_i \sum_j \frac{q_i q_j}{r_{ij}}$$

van der Waals term Electrostatic term

Equation 27: Non-bonded Energy Equation

where, **A** can be obtained from atomic polarization measurements or quantum mechanical calculations, **B** is derived from crystallographic data of observed contact distances between different kinds of atoms, and the **electrostatic term** is modeled using a Coulombic potential [30].

The **A** and **B** parameter control the depth and position of the potential energy well for a given pair of non-bonded interacting atoms (e.g. C:C, O:C, O:H, etc.) [31]. In effect, the **A** term determines the degree of stickiness of the van der Waals attraction, and **B** determines the degree of hardness of the atom (i.e. marshmallow-like, billiard ball-like, etc.) [30].

B.2.3.1.1.4 Torsion Energy

Torsion energy is primarily used to correct the remaining energy terms rather than to represent a physical process or molecular property [30]. The torsion energy equation represents the amount of energy that must be added/subtracted from the

Bond Stretching Energy + Angle Bending Energy + Non-bonded Energy Equation

terms to make the total energy agree with experimental or quantum mechanical calculations for a model of dihedral angles.

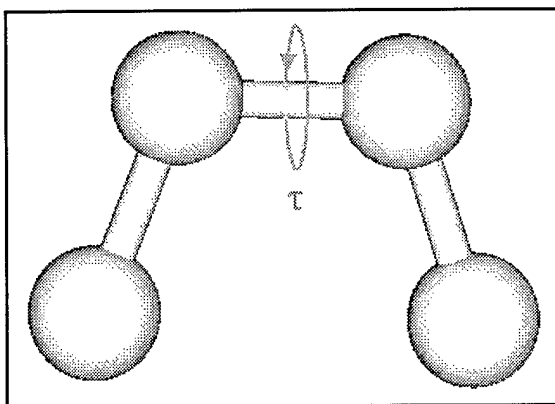


Figure 59: Torsion Energy

$$E = \sum_{\text{torsions}} A [1 + \cos(n\tau - \phi)]$$

**Equation 28: Torsion Energy
Equation**

Unique parameters for torsion rotation are assigned to each type of bonded quartet of atoms based upon their types (e.g. C-C-C-C, C-O-N-H, N-C-C-N, etc.) [30].

B.2.3.2 Semi-Empirical

Semi-empirical methods are rooted in quantum chemistry, which describes molecular energies in terms of explicit interactions between electrons and nuclei.

Quantum mechanics methods are based on the following assumptions [33]:

- 1) Nuclei and electrons are distinguishable from each other.
- 2) Electron-electron and electron-nuclear interactions are explicit.
- 3) Interactions are governed by nuclear and electron charges (i.e. potential energy) and electron motions (i.e. kinetic energy).
- 4) Interactions determine the spatial distribution of nuclei and electrons and their energies.

The theoretical foundation of quantum chemistry starts with de Broglie's [31, 33] concept (sub-atomic particles display wave-like properties), but it was Schrodinger that made the connection between classical waves and de Broglie's particle waves [33]. Schrodinger used the concept of a standing wave to quantitatively describe particle waves. The mathematical description of this wave is called a *wavefunction*.

Properties of a wavefunction describe the kinetic and potential energies of an electron in a region of space surrounding the nucleus. These properties are obtained by applying a Hamiltonian operator to the wavefunction. This generates the wavefunction (Ψ) and its corresponding energy (E). Schrodinger's equation (see **Equation 29**) can

be solved for Ψ and E . Schrodinger's equation addresses "where are the electrons and nuclei of a molecule in space?" and "what are their energies?" [33].

$$\nabla^2 \Psi = -(2\pi/[h/2m(E - V)^{1/2}])^2 \Psi = -(8\pi^2 m/h^2)(E - V)\Psi$$

Equation 29: Schrodinger's Equation

This can be rearranged by a series of algebraic steps into:

$$\begin{aligned} (-h^2/8\pi^2 m) \nabla^2 \Psi &= (E - V)\Psi \\ (-h^2/8\pi^2 m) \nabla^2 \Psi &= E\Psi - V\Psi \\ (-h^2/8\pi^2 m) \nabla^2 \Psi + V\Psi &= E\Psi \\ [(-h^2/8\pi^2 m) \nabla^2 + V]\Psi &= E\Psi \\ \text{Let } H &\equiv [(-h^2/8\pi^2 m) \nabla^2 + V] \end{aligned}$$

Equation 30: Reduction of Schrodinger's Equation

where h is Planck's constant, m is the mass of the electron, ∇ is the Laplacian operator

$$(i.e. \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}), \text{ and } V \text{ is the potential energy of the electron [34].}$$

Resulting in the following simplified form, which the Schrodinger's Equation is usually represented as:

$$H\Psi = E\Psi$$

Equation 31: Simplified Schrodinger's Equation

where Ψ is the wavefunction and E is the total energy of the electron.

Schrodinger's Equation works for hydrogen and hydrogen-like atoms, but when two or more electrons are in the atom's valance shell they not only interact with the protons in the nucleus, but also other electrons [34]. This equation becomes more complex for describing a multi-electron atom due to the electron-electron interactions (electron correlation) and an additional property called "electron spin" [33, 34, 35], but the effects can be approximated under the assumption that each electron-nuclear

interaction is screened by the average of all electrons. This leads to Schrodinger's Equation for a molecule:

$$\begin{array}{c}
 \text{\textit{T}}_{\text{electron}} \qquad \qquad \qquad \text{\textit{V}}_{\text{electron-nucleus}} \\
 H = [(-\hbar^2/8\pi^2 m_e) \sum_{i=1,k} \nabla^2 \quad - \quad \sum_{j=1,N} \sum_{i=1,k} Z_j/r_{ji}] \\
 \\
 + \sum_{i=1,k-1} \sum_{l=1+1,k} 1/r_{il} \quad + \quad \sum_{j=1,N-1} \sum_{m=j+1,N} Z_j Z_m/R_{jm} \\
 \text{\textit{V}}_{\text{electron-electron}} \qquad \qquad \qquad \text{\textit{V}}_{\text{nucleus-nucleus}}
 \end{array}$$

Equation 32: Schrodinger's Equation for a Molecule

This equation is constructed from a set of one-electron wavefunctions contributed by each atom. This approximation technique considers the nuclei to be stationary relative to the motions of the electrons [33]. The major difference between computational methods that use Schrodinger's Equation as a basis for calculating the molecular energies pertains to their consideration of the electron correlation [33].

B.2.3.3 Ab Initio

The difference between Ab Initio methods and empirical methods is that ab initio methods use the complete form of the Fock operator to construct the wave equation [30]. The decision to use the complete Fock operator makes this form of calculations computational impractical except for when dealing with the smallest of molecules. The Fock-operator is presented here for completeness (**Equation 33**) [32]. It is expressed in terms of the one-electron Hamiltonian \hat{h} (equation), the Coulomb operator \hat{J}_α (equation), and the exchange operator \hat{K}_α (**Equation 34**).

$$\hat{f}(\mathbf{r}_1) = \hat{h}(\mathbf{r}_1) + \sum_{\alpha=1}^{N/2} (2\hat{J}_\alpha(\mathbf{r}_1) - \hat{K}_\alpha(\mathbf{r}_1))$$

Equation 33: Fock Operator

$$\begin{aligned}
 h(\mathbf{r}_1) &= -\frac{1}{2}\nabla_1^2 - \sum_{A=1}^M \frac{Z_A}{|\mathbf{r}_1 - \mathbf{R}_A|} \\
 J_a(\mathbf{r}_1) &= \int d\mathbf{r}_2 \psi_a^*(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \psi_a(\mathbf{r}_2) \\
 K_a(\mathbf{r}_1) \psi_i(\mathbf{r}_1) &= \left[\int d\mathbf{r}_2 \psi_a^*(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \psi_i(\mathbf{r}_2) \right] \psi_a(\mathbf{r}_1)
 \end{aligned}$$

Equation 34: Hamiltonian, Coulomb, and Exchange Operators

The minimum energy can be calculated using the $N/2$ spatial orbitals with the lowest eigenvalues $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_{N/2}$. However, the total electronic energy is not just the sum of these $N/2$ eigenvalues. The correct expression for the energy is

$$E_0 = 2 \sum_{a=1}^{N/2} \langle \psi_a | h | \psi_a \rangle + \sum_{a,b=1}^{N/2} \left[2 \langle \psi_a \psi_b | \frac{1}{r_{12}} | \psi_a \psi_b \rangle - \langle \psi_a \psi_b | \frac{1}{r_{12}} | \psi_b \psi_a \rangle \right]$$

Equation 35: Minimum Energy Equation

where the terms in $\langle \rangle$ are defined as:

$$\begin{aligned}
 \langle \psi_a | h | \psi_b \rangle &= \int d\mathbf{r}_1 \psi_a^*(\mathbf{r}_1) h(\mathbf{r}_1) \psi_b(\mathbf{r}_1) \\
 \langle \psi_a \psi_b | \frac{1}{r_{12}} | \psi_c \psi_d \rangle &= \iint d\mathbf{r}_1 d\mathbf{r}_2 \psi_a^*(\mathbf{r}_1) \psi_b^*(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \psi_c(\mathbf{r}_1) \psi_d(\mathbf{r}_2)
 \end{aligned}$$

Equation 36: Definition of Matrix Elements

The problem of finding solutions for this time independent Schrodinger equation is now reduced to finding solutions of the eigenvalue equation called the "Hartree-Fock" equation (**Equation 37**).

$$f(\mathbf{r}_1) \psi_j(\mathbf{r}_1) = \varepsilon_j \psi_j(\mathbf{r}_1)$$

Equation 37: Hartree-Fock Equation

The only remaining problem with using **Hartree-Fock Equation** to solve for the conformation state of a protein is that it has an infinite number of solutions [32]. Therefore, the next step is to expand the spatial orbitals in a finite set of known basis functions:

$$\psi_i = \sum_{\mu=1}^K C_{\mu i} \phi_{\mu} \quad i = 1, \dots, K \quad K \geq N/2$$

Equation 38: Spatial Orbitals

The orbitals $N/2+1 \dots K$ are the “unoccupied” two-electron states [32]. The coefficient $C_{\mu i}$ is unknown and still has to be determined. A straightforward application of this expression to the **Hartree-Fock Equation** leads to a system of algebraic equations called the **Roothaan Equations**:

$$\mathbf{F} \mathbf{C} = \mathbf{S} \mathbf{C} \boldsymbol{\epsilon}$$

Equation 39: Roothaan Equations

All the symbols in this equation are $K \times K$ matrices. The matrix $\boldsymbol{\epsilon}$ is a diagonal matrix with values $\epsilon_1 \dots \epsilon_K$. The matrix \mathbf{C} has the elements $C_{\mu i}$. The “overlap matrix” (\mathbf{S}) and the “Fock matrix” (\mathbf{F}) have elements defined by

$$\begin{aligned} S_{\mu\nu} &= \int d\mathbf{r}_1 \phi_{\mu}^*(\mathbf{r}_1) \phi_{\nu}(\mathbf{r}_1) \\ F_{\mu\nu} &= \int d\mathbf{r}_1 \phi_{\mu}^*(\mathbf{r}_1) f(\mathbf{r}_1) \phi_{\nu}(\mathbf{r}_1) \end{aligned}$$

Equation 40: Overlap and Fock Matrices

The expressions for $\mathbf{F}_{\mu\nu}$ can be expanded, but it still leaves us with the problem of solving a system of nonlinear equations. This non-linearity arises because the Fock operator depends upon the coefficients $\mathbf{C}_{\mu i} (\mathbf{F} = \mathbf{F}(\mathbf{C}))$. The only possible way to solve this equation takes the form of iterating the equations until a solution to **Equation 35**:

Minimum Energy Equation does not change within some specified accuracy between two successive iterations, but convergence is not guaranteed [32]. The order of complexity for this set of equations is determined by the calculation of the two-electron

integrals $\left\langle \phi_{\mu} \phi_{\sigma} \left| \frac{1}{r_{12}} \right| \phi_{\nu} \phi_{\lambda} \right\rangle$, and will generally consume the most processor time because

of their large number ($\Theta\left(\frac{K^4}{8}\right)$ unique integrals) [32].

The whole process is summarized as [32]:

- 1) Write down the Schrodinger equation for the system.
- 2) Use Slater determinant, containing molecular orbitals as the wave function.
- 3) Obtain the nonlinear Hartree-Fock equations by use of the variational principle.

- 4) Introduce a finite basis set to obtain the algebraic equations.
- 5) Try to solve these equations using an iterative approach.

B.3 Summary

The three computational methods empirical, semi-empirical, and ab initio vary in their ability to accurately model a molecule at the atomic level., but other just as important properties of these three methods must be considered prior to choosing a particular method for inclusion into a computational engine for solving the Protein Structure Prediction Problem. The “practical” differences are listed in **Table 1**.

Appendix C: Parallelization Techniques

Ever since conventional serial computers were invented, their speed has steadily increased to match the needs of emerging applications. However, as we approach the fundamental physical limitation of a serial computer imposed by the speed of light, it is increasingly costly and difficult to achieve further improvements in the speed of a single processor computer [37]. Therefore, more and more scientists have turned to parallel computers in the hopes for faster execution of computationally intensive applications.

A major stumbling block in parallel computing is the difficulty in conceptualizing parallel approaches to problem solving. People tend to think of problem solutions in a sequential fashion, but sequential (serial) solutions to problems rarely transform into quality parallel solutions. There are two major concerns when parallelizing any algorithm:

- 1) Is the parallel algorithm correct? (Effectiveness)
- 2) Is the parallel algorithm faster than the serial version? (Efficiency)

Correctness is an issue because there is greater difficulty in verifying correctness of parallel algorithms than sequential algorithms [36, 37]. If we assume the algorithm is correctly implemented, then speedup becomes the primary issue and goal of parallelization. A trade-off analysis is generally required to determine if the estimated benefits warrant the expenditure of resources to parallelize an algorithm. There are several different techniques and software utilities that can help us understand parallelization trade-offs as we develop parallel applications.

C.1 Decomposition Techniques

Data and control decomposition are alternate means to dividing a serial algorithm into portions that can be performed simultaneously. In general, data decomposition allows for *data parallelism*, and control decomposition enables a parallel programmer to parallelize the control of an algorithm (*control parallelism*). In data parallelizable algorithms, many data items are subject to identical processing. Assigning data elements to various processors, each of which performs identical computations on its data, parallelizes such problems [36]. On the other hand, control parallelism refers to the simultaneous execution of different instructions. These types of

parallelized programs can either be executed on the same data stream or on different data streams [36]. In either case (data or control decomposition) the results are combined in some fashion to obtain the final solution.

Genetic algorithms (GA) are highly data parallelizable. Parallelizing a GA is as simple as running multiple copies on separate populations (using a different random seed) and processors then choosing the best result from all the runs. Data parallelization techniques are also amenable to static load balancing because their computation and communication patterns are regular [15]. Historically, AFIT has data decomposed the Protein Structure Prediction (PSP) problem, but this is not to say control parallelism is impossible.

Several years ago, Charles Brooks and Bill Young developed a heterogeneous version of CHARMM (a computational engine used in the search for an answer to PSP problem) which tackled the intense computational demands of simulating a protein surrounded by water [38]. This approach took advantage of the data parallelism nature of the problem (i.e., computations for the water molecules are independent of the others) [38]. Their newest implementation of CHARMM is a distributed version executing on the CRAY T3D and C90 (coupled). This version has already shown two to three times the speed-up over previous implementations, which were hampered by communications overhead [38]. Furthermore, this version not only takes advantage of data decomposition it also takes advantage of task decomposition by assigning the water molecule interactions to the C90 vector supercomputer [38] and using pipelining principles.

C.2 Scheduling Strategies

Once we have decomposed our algorithm into “manageable” tasks, we have to schedule these tasks for execution on a particular computer architecture. This scheduling problem boils down to resource allocation decisions consisting of *placement* and *assignment*. Placement is simply defined as – “*where to locate code and data in physical memory* [43]?” Assignment, on the other hand, tries to answer the question of “*which processor will execute each task* [43]?” There are two general ways to answer these questions: static scheduling and dynamic scheduling.

Static scheduling assigns the tasks to the processors prior to program execution using task weights and processing resources. The tasks will always execute on the

processor on which they were assigned [43]. Of course, the quest for an “optimum” schedule is NP-complete. However, there are several sub-optimal techniques that have been shown to work quite well. Alas, static scheduling is not as portable as dynamic scheduling nor is it guaranteed to work on the same architecture over time if the platform’s configuration is unstable.

Dynamic scheduling, on the other hand, is best when very little prior knowledge is available about the resource needs of the tasks and when we are not sure where the program will execute during its lifetime. Dynamic schedules are either adaptive or non-adaptive. Adaptive schedules change dynamically in response to shifting system loading [43]. Non-adaptive schedules do not. Even though they support greater portability across architecture platforms, there is an associated cost. Dynamic scheduling entails execution time overhead cost(s) for determining the schedule. A trade-off analysis is called for to determine which scheduling strategy would be best for any particular algorithm instantiation.

C.3 Load Balancing

Load balance goes hand in hand with scheduling strategies. In load balancing, we are trying to distribute the workload from the heavily loaded processors to the lightly loaded processors with the purpose of improving the overall performance of the system [43]. Load balancing algorithms consist of three components: *information policy*, *transfer policy*, and *placement policy*. The information policy specifies the amount of load and task information made available to the task placement decision maker(s) and the way this information is distributed [43]. In short, we are answering the question of “*how do we know the task load has become unbalanced?*” The second component determines the suitability of a job for load transferring. The transfer policy is trying to answer the question of “*which task will we transfer?*” It is usually based on the load of the host processor and the size of the task [43]. The third and final component – the placement policy – answers the question of “*where do we put the transferring task?*” There are many placement schemes (e.g., round robin, closest neighbor, least loaded, etc.).

Load balancing can either be accomplished prior to program execution – static load balancing – or during program execution – dynamic load balancing. There are many ways to accomplish either static load balancing or dynamic load balancing. A

trade-off analysis should be accomplished to determine which strategy best suits your program and target architecture. Static load balancing tends to be easier to implement (if enough information is available a priori), and in some cases it can achieve an “optimal” balancing. On the other hand, dynamic load balancing performs better when the characteristics of the program or the topology of the target computer architecture change significantly over time in a way we cannot easily predicate, but we will pay an overhead price for dynamically balancing the program load. **Appendix D** looks at those scheduling and load balancing schemes we have analyzed for the problem-algorithm integration.

C.4 Introduction to UNITY

Chandy and Misra proposed an architecture independent method for the description of an algorithm – UNITY (Unbounded Nondeterministic Iterative Transformations) [39]. A UNITY specification describes the requirements for a process not the “how” of the process [39]. The UNITY approach embodies three important concepts:

- 1) high-level, explicit expression of parallelism,
- 2) extrication of proofs from the basic program design, and
- 3) mapping of the initial design to a specific parallel architecture while maintaining correctness.

C.4.1 Explicit Expression of Parallelism

The UNITY approach isolates the program designer from the specifics of a given parallel architecture. The resulting description is strong on “what,” and says practically nothing about “when” or “how” [39]. Design decisions forced by the target architecture are postponed until late in the design process. This approach helps the designer extract all the inherent parallelism of an algorithm and to implicitly define decomposition options [39]. A high-level UNITY specification is written as a series of assignment statements capable of being executed in parallel. Using this product, the designer can then examine in detail the complexity of each independent piece of the program.

Table 37 discusses the principles which form the basis of parallelism within a UNITY design [39]:

- **Non-Determinism** – A UNITY specification is a set of executable assignment statements, and under this concept each statement is executed infinitely often (The Fairness Rule).
- **Absence of Control Flow** – UNITY specifications do not specify control flow. Divorcing control flow from program construction allows for greater flexibility in mappings to different parallel architecture.
- **Synchrony and Asynchrony** – The UNITY language supports synchronous and asynchronous assignment of variables.
- **States and Assignments** – All UNITY specifications are composed entirely of states and assignments. Progression from state-to-state takes place through parallel or sequential assignment of variables. This principle supports the extraction of proofs from UNITY specifications.

Table 37: Principles of UNITY

C.4.2 Extraction of Proofs

Using a combination of standard logic operators and special temporal logic operators, a proof of program correctness is extracted from the high-level UNITY specification. This is the power of the UNITY design approach. This is a great step forward in answering the first question proposed earlier (Section Error! Reference source not found.): *Is the parallel algorithm correct?* The ease of proving the UNITY design depends on how well the design is written and the level of detail it contains.

Typically, the UNITY design proof proceeds in the following manner [39]:

- 1) Define and prove the existence of an *Invariant* (something that is always true about the design throughout its execution).
- 2) Define and prove the existence of a Fix Point. The Fix Point is the stopping condition.
- 3) Define a progress property and show the progress property holds until the fix point is reached.

Table 38: Steps to a UNITY Proof

C.4.3 Mapping of the Initial Design

The UNITY approach provides for a means of transforming a high-level specifications into an intermediate forms using correctness-preserving mappings [39]. The intermediate form provides a structure for developing an implementations on a

specific parallel architectures, whether it is sequential, asynchronous shared-memory, or distributed, while maintaining the programs correctness. Source code for the executable program can be written based loosely on the intermediate form [39]. The description of the mappings describes “how” the UNITY program is executed on the target machine. Mappings for particular classes of architectures exhibit common characteristics [14].

Chandy and Misra provide the following mapping strategies for asynchronous shared-memory architectures (**Table 39**), distributed architectures (**Table 40**), and synchronous architectures (**Table 41**) [40]:

<p>The mapping strategy of a UNITY design to asynchronous shared-memory architectures is as follows:</p> <ol style="list-style-type: none"> 1) Allocates each statement in the program to a processor, 2) Allocates each variable to a memory location, and 3) Specifies the control flow for each processor. <p><i>And this mapping must satisfy the following constraints:</i></p> <ul style="list-style-type: none"> • All variables on the left side of each statement allocated to a processor are in memory that can be written to by the processor, and all variables on the right side are in memories that can be read by the processor. • The control flow for each processor is such that every statement allocated to the processor is executed infinitely often.

Table 39: Mapping to Asynchronous Shared-Memory Architectures

The mapping strategy of a UNITY design to **distributed architectures** is the same as for asynchronous shared-memory architectures except:

- 1) Each variable is allocated either to the local memory of a processor or to a channel.

The mapping must satisfy the following constraints in addition to the constraints of the shared-memory case:

- At most one variable is allocated to each channel, and this variable is of type sequence.
- A variable allocated to a channel is named in statements of exactly two processors, and these statements are of the following form: The statements in one of the processors modify the variable by appending an item of data (the message) to the rear of the sequence, if the size of the sequence does not exceed the buffer size; statements in the other processor modify the variable by deleting the items at the head of the sequence, if the sequence is not null. The variable is not accessed in any other way.

Table 40: Mapping to Distributed Systems

In general, mapping a UNITY design to **synchronous architectures** is complex. Therefore, I will restrict this discussion to what the mapping should consist of:

- A description of how the operations in each statement are to be executed by the processors,
- An allocation of each variable to the memory, and
- A specification of a single flow of control, common to all processors.

And this mapping must satisfy the following constraints:

- The manner in which processors execute a statement is consistent with the allocation of the variables to memories.
- The flow of control must be such that each statement is executed infinitely often.

Table 41: Mapping to Synchronous Architectures

Appendix D: Decomposition of the PSP Problems

D.1 Decomposition of PSP Problem

Many different genetic algorithms have been used to manipulate the search space in AFIT's continual efforts to find the global optimum conformational energy for general proteins. We have added to these approaches by incorporating the problem domain into the Linkage Learning GA (LLGA) developed by Harik [7] and incorporating the CHARMM energy model [41, 42] as their primary fitness functions. The LLGA's pseudo-algorithm is in **CHAPTER 3**. LLGA's processes are broken into the tasks identified in **Table 42**.

Task Number	Description	Average Execution Time	Workload
1	Initialization – creation of initial population	4	4
2	Fitness Evaluation – CHARMM	5	5
3	Tournament Selection	2	2
4	Exchange Operation	3	3
5	Stopping Condition – determines if objective is reached and records final most fit individual	1	1

Table 42: LLGA Task Decomposition

We have ranked the task with respect to average execution time and workload (1 being the lowest value and 5 is the highest).

Since, we are primarily comparing the LLGA to the fast messy GA (fmGA) in **CHAPTER 5** and **CHAPTER 6**, we have presented the corresponding task decomposition in **Table 43**. The pseudo-algorithm is in **CHAPTER 3**.

Task Number	Description	Average Execution Time	Workload
1	PEI – creation of initial population	7	6
2	Fitness Evaluation – CHARMM	8	8
3	Tournament Selection	4	5
4	Reduce Population Size	5	4
5	Primordial Phase Stopping Condition – record population	3	3
6	Cut-And-Splice	6	7
7 ^{xxx74}	Juxtapositional Stopping Condition – determines if objective is reached and records K th -order Template	2	2
8	Stopping Condition – determines if objective is reached and records final most fit individual	1	1

Table 43: mGA Task Decomposition

Both of these decompositions (Table 42 and Table 43) lead to data decomposed solutions. An alternative method towards decomposing this problem leads to parallelization of the CHARMM energy model (i.e., a task decomposition model). The decomposition of the CHARMM energy model as implemented at AFIT follows (see SECTION):

Task Number	Description	Average Execution Time	Workload
1	Bond Energy	6.00e-6 sec	2
2	Angle Energy	5.00e-6 sec	3
3	Torsion Potential	1.04e-3	5
4	Improper Torsion	too small to measure	4
5	Lennard-Jones Equation	2.33e-2 sec	6
6	Energy Constant	too small to measure	1

Table 44: CHARMM Decomposition

In order to implement the decomposition presented in Table 44, it would require passing a message containing the complete molecule layout to each subtask in order for them to compute the total energy. The size of this message is approximately 240 bits (representing the protein [Met]-Enkephalin) plus 4,700 bits (molecule layout represented in a PDB file). Therefore, this model of parallelization was not implemented nor will it be discussed further.

⁷⁴ I have combined the stopping condition check with saving the kth-order template.

D.2 Scheduling and Load Balancing of the GAs

Just looking at the benefits of the LLGA and fmGA decomposition schemes presented in **Figure 60** and **Figure 61**, we can tell that little parallelization is possible since each processor must communicate its results to the next processors.

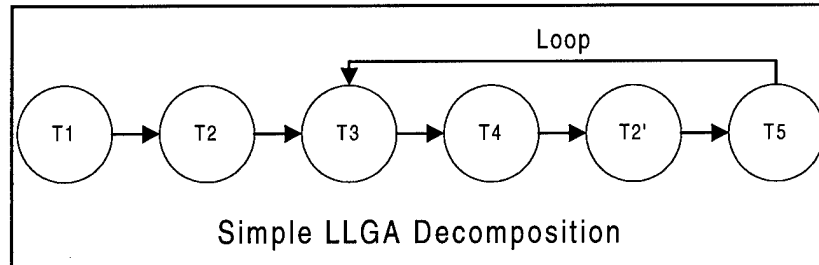


Figure 60: Direct LLGA Task Decomposition Schedule

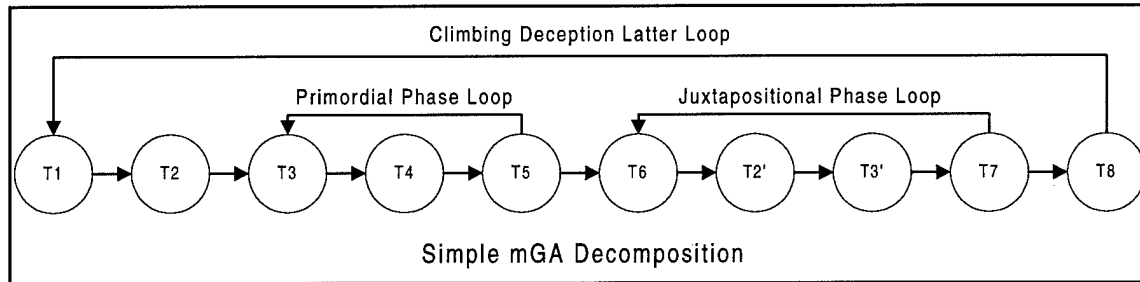


Figure 61: Direct fmGA Task Decomposition Schedule

The best possible scheme may come from combining tasks onto single processor, which would cut the communication overhead considerably and more evenly distribute the workload. This sort of scheduling follows from Zhu's workload scheduling [43]. In Zhu's algorithm, the scheduling is accomplished by computing the workload associated with each level of a directed acyclic graph (DAG) starting with the source node. Neither of the above graph is a DAG. We have chosen to ignore this constraint for the time being because we could unfolded the cycles in each of the above graphs by simply duplicating the looped tasks an "appropriate" number of times (i.e., analagously to compiler "loop-unrolling"). The next step involves determining whether a task should be aggregated with its predecessor or whether it should be parallelized (see **Figure 62** and **Figure 63** for result of aggregation). Once we have developed this new set of DAGs, we can compute the workload for each processor, and once we have aggregated these workload values up the DAG, we can determine the final schedule for completing the

task. The DAGs indicate a compression of communication and computation and a possible final task scheduling scheme for these GAs.

When developing the new DAG for the fmGA (**Figure 63**) and LLGA (**Figure 62**), we considered the expected execution time and workload aggregated as a single component. In order to represent this, we assumed that each ordinal unit represents a single time unit. For example, if we looked at task 1 defined in **Table 42**, we see task 1 defined as having 2 for expected execution time and 2 for workload. Therefore, the task would have a computation value of 4. The communication times for each task are based on the amount of data each task receives from the preceding processor and sends to the next processor. The sizes of the messages are based on the size of the population. Therefore, in the DAGs, we have indicated the portion of the population needing to be passed to the next processor. If the whole population needs passing, I have indicated this with a “p.”

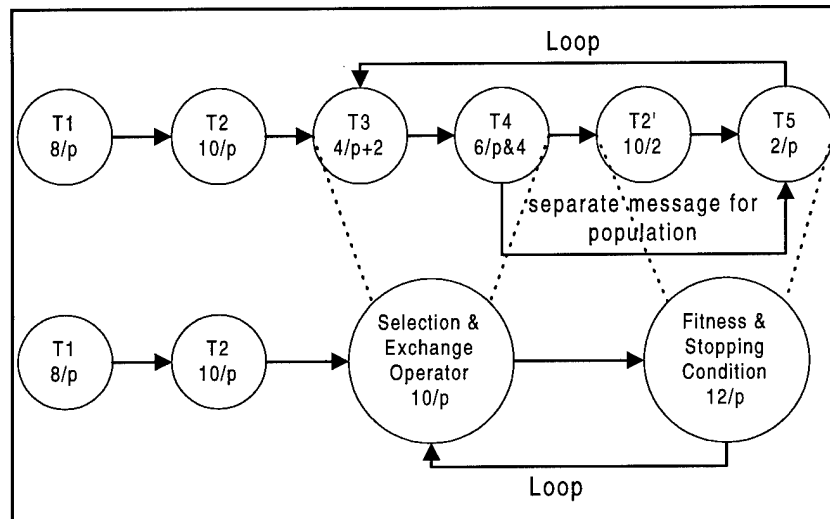


Figure 62: Zhu's Scheduling for LLGA

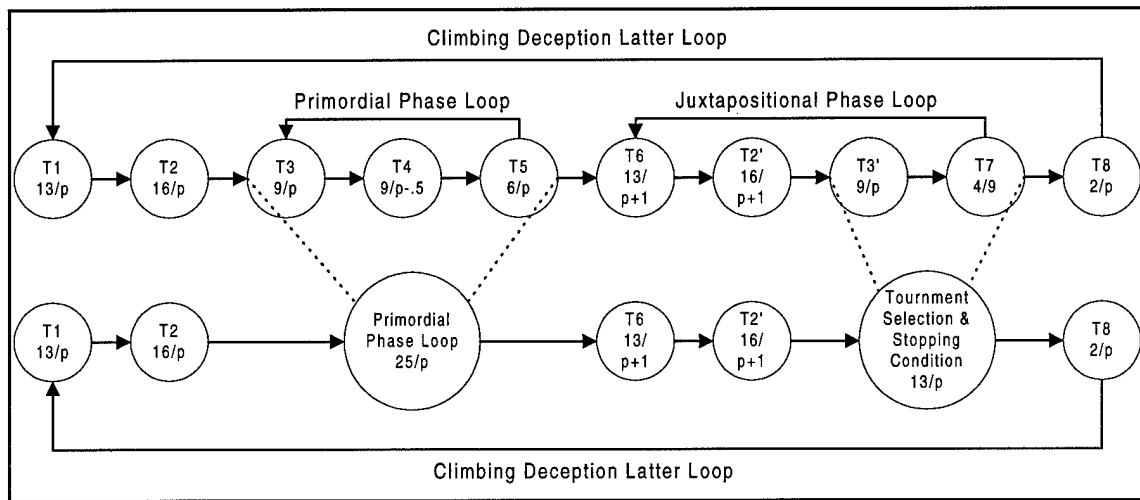


Figure 63: Zhu's Scheduling for fmGA

An alternative method for parallelizing the GA that we felt it was important to investigate is Kruatrachue-Lewis' Duplicate scheduling [43]. The basic idea is to duplicate task to reduce communication delays, at the cost of increasing the space complexity and total computation of the program. This paradigm is similar to the "Farming Model" or "Island Model" presented by Gates [15] and Kaiser [37]. This sort of scheduling strategy leads to the following DAGs (see **Figure 65** and **Figure 64**). We've based the task duplication on the same computation and communications estimates developed for Zhu's algorithm.

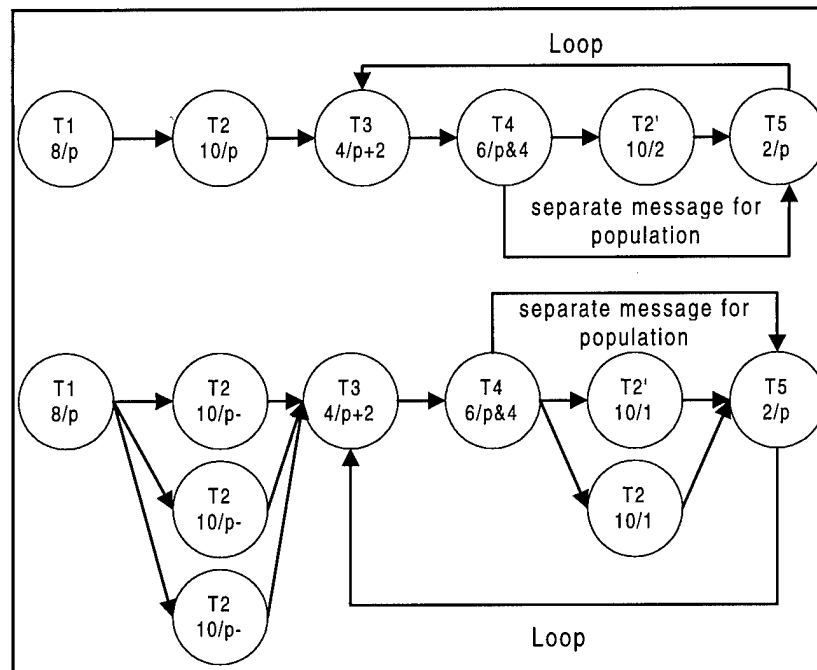


Figure 64: Kruatrachue-Lewis' Duplicate Scheduling for LLGA

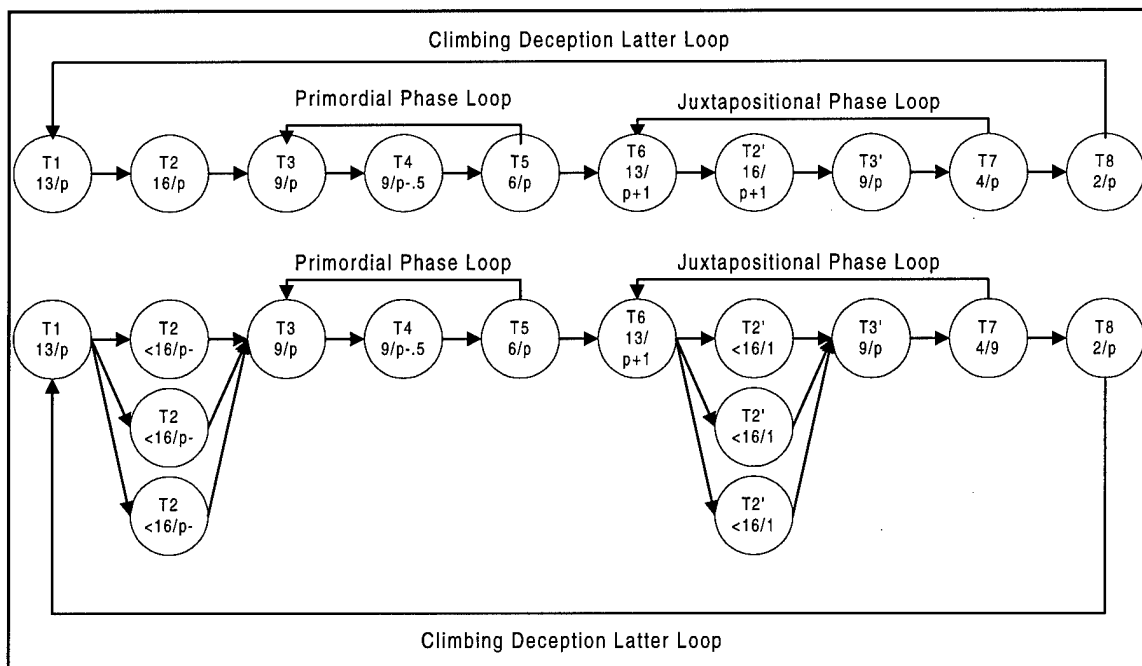


Figure 65: Kruatrachue-Lewis' Duplicate Scheduling for mGA

D.3 Scheduling and Load Balancing of CHARMM

Turning our attention to the decomposition of CHARMM, we again see little gains by blindly parallelizing the equation across several processors, but after applying the two aforementioned scheduling schemes we can see appreciable performance improvements. Even though the calculated gain is not significant for the small molecule trial figures, we argue that by parallelizing the CHARMM model, we will see huge dividends when we attempt to “CHARMM-menize” a much larger protein.

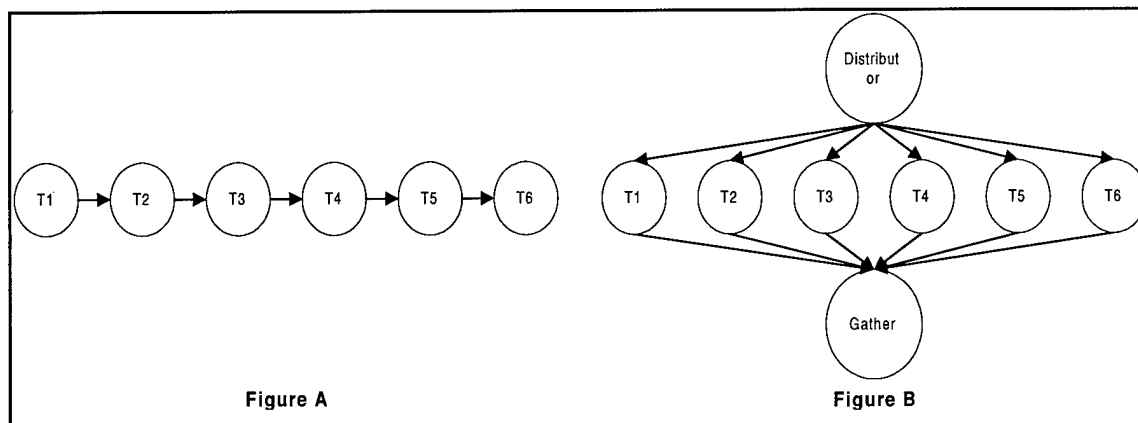


Figure 66: Two Simple Parallelizations of CHARMM

First let us look at **Figure 66**. This figure presents two distinct ways to graph CHARMM as a DAG. In both figures, we have scheduled each task on separate processors, but **Figure B** affords us a greater possibility for parallelization. **Figure A** is a linear pipeline model in which the intent is to pass a single chromosome through at a time in the hopes of achieving some speed-up. This model may achieve some speed-up over “normal, non-decomposed” CHARMM, but it has a serious bottleneck at Task 5 (T5) that severely hampers the potential gain. The bottleneck is similar to float-point division bottleneck found in CPU design (i.e., the fastest the **Figure A** CHARMM pipeline can go is determined by the Lennard-Jones Equation). On the other hand, **Figure B** circumvents this problem by allowing out-of-order completion of each chromosome’s fitness evaluation even though we have the added overhead of distribution and gather of the final answer. Out-of-order completion could be implemented by allowing the “gather process” to know a priori the population size and implementing an in order traversal for distributing the population. This way the gather progress would understand that the first message received by any particular task would

belong to the first member of the population and so on and so forth (i.e., the second message from the same task would go to the second member of the population).

At this time, we should emphasize the serious shortcoming to any decomposition of CHARMM: the required message size. The message size is a limiting factor because for a relative small protein model (for instance [Met]-Enkephalin) the file size is approximately 4,700 bits which would need to accompany each chromosome through the CHARMM calculation process. We propose a simple yet elegant method to get around this bottleneck. The file is static; it never changes. Therefore, we could have each processor load the file as part of its initialization while the other tasks of the GA are accomplishing their work. Therefore, the message size would be reduced to the length of the chromosome plus the length of the energy result.

Zhu's scheduling for CHARMM is in **Figure 67**.

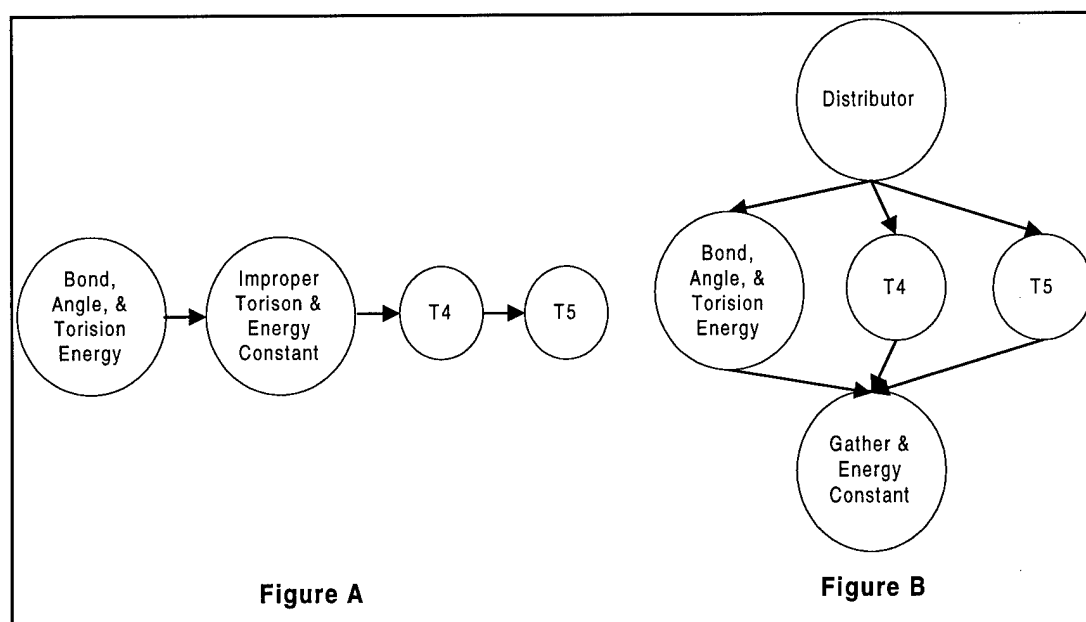


Figure 67; Zhu's CHARMM Schedule

Figure A provides the overall greatest return because it is not hampered by have the Distributor or Gather processes of **Figure B** which are overhead.

Kruatrachue-Lewis' Duplicate scheduling scheme produces the DAGs found in **Figure 68** which show that again the linear pipeline looks as if it will provide the greatest improvement because it does not have the additional overhead.

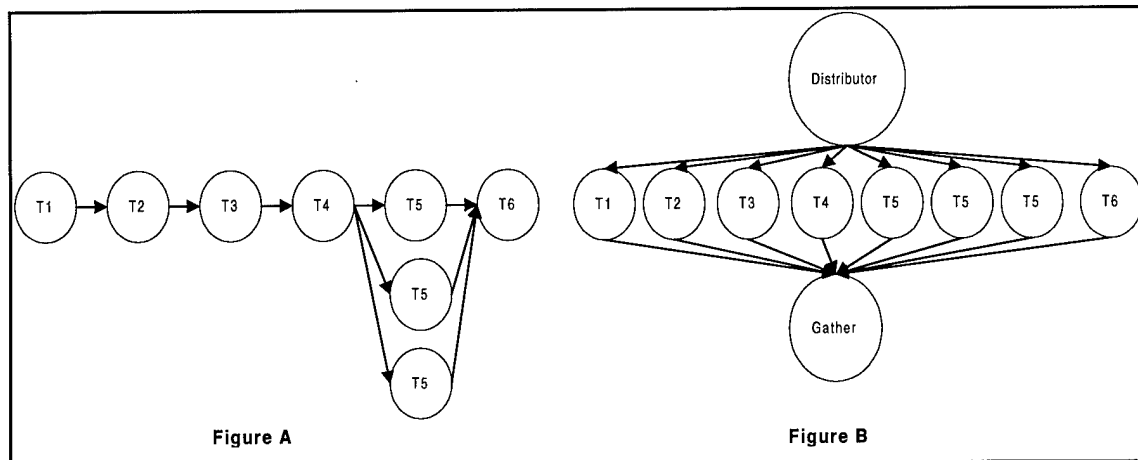


Figure 68: Krutrachue-Lewis' CHARM Schedule

D.4 Summary

In this appendix, we have investigated several different possibilities for parallelizing the LLGA and the CHARMm energy model in order to determine which methodology produces the greatest speed-up. CHARMm requires an average 1.09 seconds to evaluate each chromosome. Thus, a 50 chromosome population, the worst case execution time for CHARMm is 54.50 seconds per generation. On the other hand, the LLGA requires approximately 4.5 seconds of processing per generation. Therefore, we follow Krutrachue-Lewis' Duplicate scheduling scheme for our parallelization methodology where we will be parallelizing the fitness evaluations.

Appendix E: Additional LIGA Structure

E.1 Messy Genetic Algorithm (mGA)

The mGA proposed by Goldberg *et al.*, in 1989, was a major paradigm shift for its time. The mGA was the first to suggest moving from “neat coding and operators” to allowing variable-length strings that may be under- or over-specified with respects to the problem being solved [1]. The original mGA was designed to handle the “deception problem.” Goldberg’s originally proposed mGA was fashioned from his view that nature’s climb out of the primordium occurred with genotypes that exhibit redundancy, over-specification, under-specification, changing length, and changing structure [1].

E.1a mGA Chromosome Representation: Positional Precedence

Positional precedence may also permit the formation of a kind of *intra-chromosomal dominance operator* [1]. This intra-chromosomal dominance operator was not used in the original mGA, but the concept is useful in large allelic alphabets. It allows the mGA user to pre-specify some precedence relationship amongst the allele values. Then, as the mGA handled over-specification, it takes into account these precedences [1]. **Figure 69** illustrates how the intra-chromosomal dominance operator works.

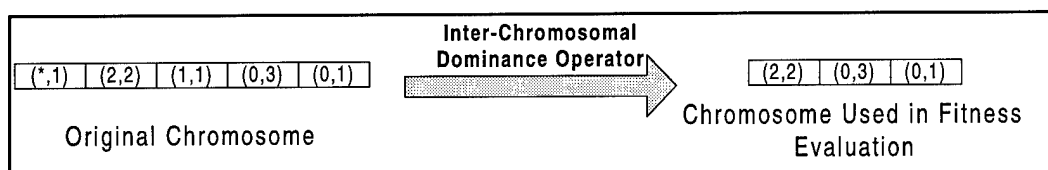


Figure 69: mGA's Inter-Chromosomal Dominance Operator

If the allelic alphabet is [0, 1, *] with the inter-chromosomal precedence [1^{st} , 2^{nd} , 3^{rd}] respectively, then the inter-chromosomal dominance operator would skip the (*, 1) and (1,1) alleles and use the (0,1) allele value in the final chromosome representation sent to the fitness function.

E.1b mGA Chromosome Representation: the Competitive Template

The key to this notion of using locally optimal template is salient building blocks. If the relative rankings of the best building block and the other building blocks are

preserved using a template, and as long as the other building blocks have a template fitness that is less than or equal to that of the lowest locally optimal point, then selection is expected to yield building blocks that are at least as good as the lowest locally optimal point [1]. Therefore, only salient building blocks obtain fitness values better than the template value [2], and according to the Schema Theorem, their representation should increase within the population. The “trick” to using a competitive template is its generation. (i.e., how can a locally optimal template be generated without the prior knowledge of the fitness landscape?) In his original discussion, Goldberg made a couple of suggestions using mathematical criteria such as linear local optimization techniques, as well as preprocessing the problem domain in SGA and using the returned “best” chromosome as a basis for the template [1].

In later papers covering the mGA, Goldberg suggests the template generation method most commonly used today – *climbing the-ladder-of-deception* [2]. This concept can be best explained as solving an order- k deceptive problem by first solving it to order- $(k-1)$ optimality and then finding the necessary order- k improvements to that solution [3]. Therefore, most mGA implementations have what is called an “outer-loop” that increments the order of the deception the mGA is trying to combat (see Algorithm 1). For example, by starting at the $k = 1$ level, a 1st-order optimal template can be found, which in turn can be used in solving for the $k = 2$ template, and so on until the k^{th} -order of deception that characterizes the problem domain in question is accomplished [2].

E.1c mGA Algorithmic Phases: partially enumerative initialization (PEI)

The building blocks created for in PEI are an exhaustive list of allele combinations of length equal to the estimated build block size or nonlinearity of the problem domain [14]. If the building block size is greater than or equal to the level of deception present in the problem, the PEI phase guarantees that all building blocks necessary to form the globally optimal solution are represented in the initial population [14]. This results in the mGA’s PEI population size being governed by **Equation 41** [1].

$$n = C^k \binom{l}{k}$$

where, k is the order of the building block,
 C is the cardinality of the alphabet,
 l is the string length, and
 $\binom{l}{k}$ is the number of combinations of k
genes.

Equation 41 : PEI Population Equation

This leads to rather large populations quickly. **Figure 70** indicates the rapid population growth rate required in PEI for a chromosome length of 240 binary alleles. Specifically, the number of building blocks required in the PEI phase for this chromosome assuming a 3rd-order deception problem is:

Using **Equation 41**: $n = C^k \binom{l}{k}$

$l = 240$	(chromosome length)
$k = 3$	(3 rd -order deception problem)
$C = 2$	(binary allelic alphabet)

Therefore,

$$n = 2^3 \binom{240}{3}$$

$$n = 8(2275280)$$

$$n = 18,202,240 \text{ initial building blocks (indicated on Figure 70)}$$

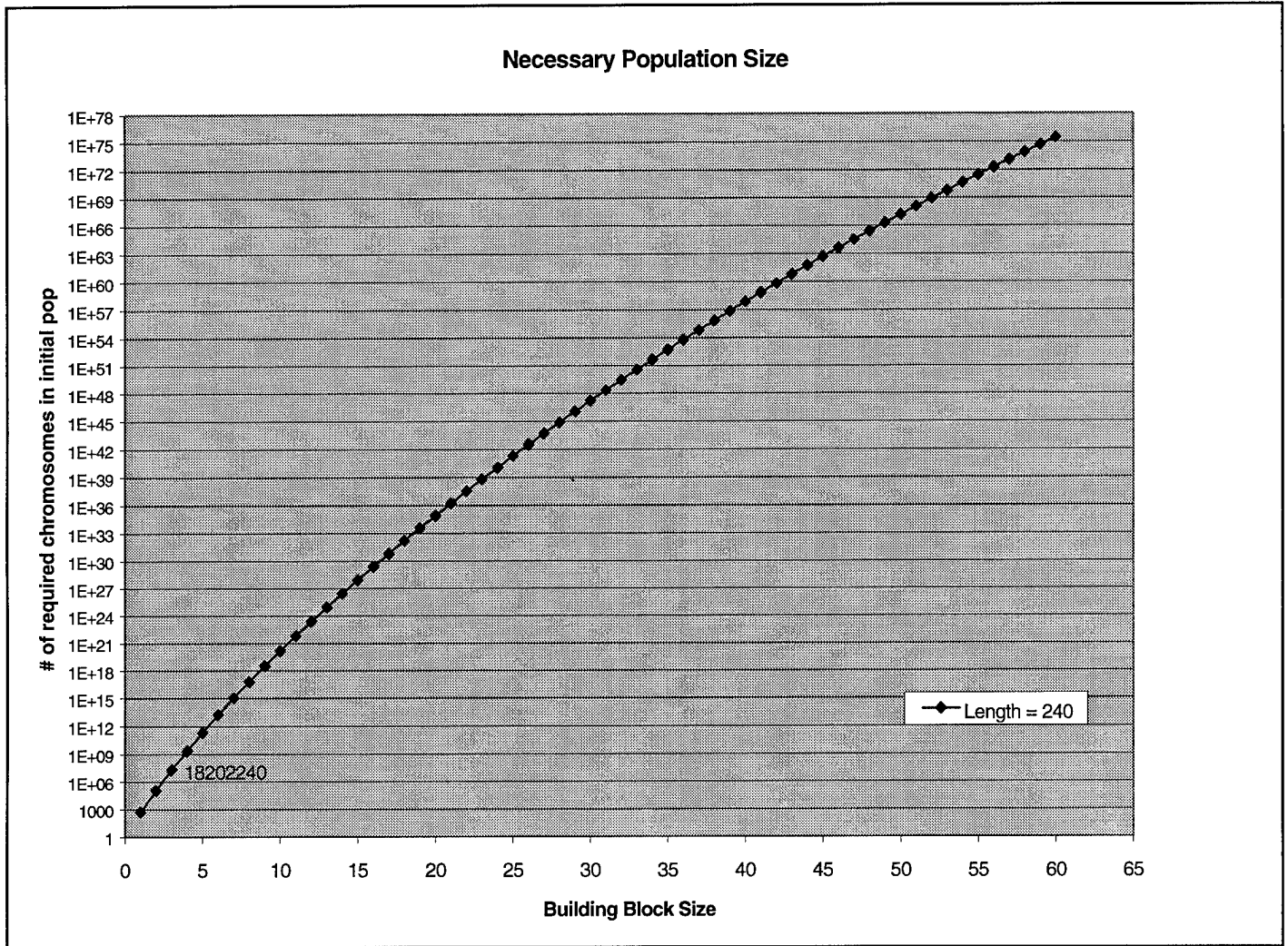


Figure 70: Population Growth Rate for the mGA

E.1d mGA Algorithmic Phases: Cut-and-Splice

Cut-and-splice can be thought of a simple one-point crossover operating on variable length strings. But, this operator was specifically designed to handle variable-length as well as over-specified and under-specified chromosomes [1]. A “cut” is performed first on each of two individual chromosomes, randomly chosen from the population, using a specified bitwise cut probability (p_k) [1, 16]. The overall cut probability for each individual is governed by **Equation 42** where λ is the current length of the string and p_c is subject to the limit $p_c \leq 1$.

$$p_c = p_k(\lambda - 1)$$

Equation 42: Cut Probability

Thereafter, the cut operator cuts an individual at a position chosen uniformly at random along its length [1] if a "cut" is dictated.

The "slice" operator concatenates the cut portions of the mating individuals in order to produce new population members. The cut chromosomes are recombined with a specified slice probability (p_s). Dymek [48] observed that there are four possible outcomes from the cut operator: 1) neither chromosome is cut, 2) only the first chromosome is cut, 3) only the second chromosome is cut, or 4) both chromosomes are cut. From these "cut cases," Goldberg's splice operator systematically checks the possibility of splicing only successive pairs [1], while Dymek points out that more complex manipulations of the cut chromosomes are possible if we don't limit the splice operator to just successive pairs [16]. **Figure 71** illustrates Goldberg's view of the cut-and-splice.

Dymek's view, although feasible, has not been implemented. His view of the cut-and-splice operator would add more splice possibilities to each case. (See **Table 45**)

CASE	Goldberg	Dymek
1	2	3
2	3	7
3	3	7
4	5	13
Total Possibilities	13	31

Table 45 : Cut-and-Splice Combination Possibilities

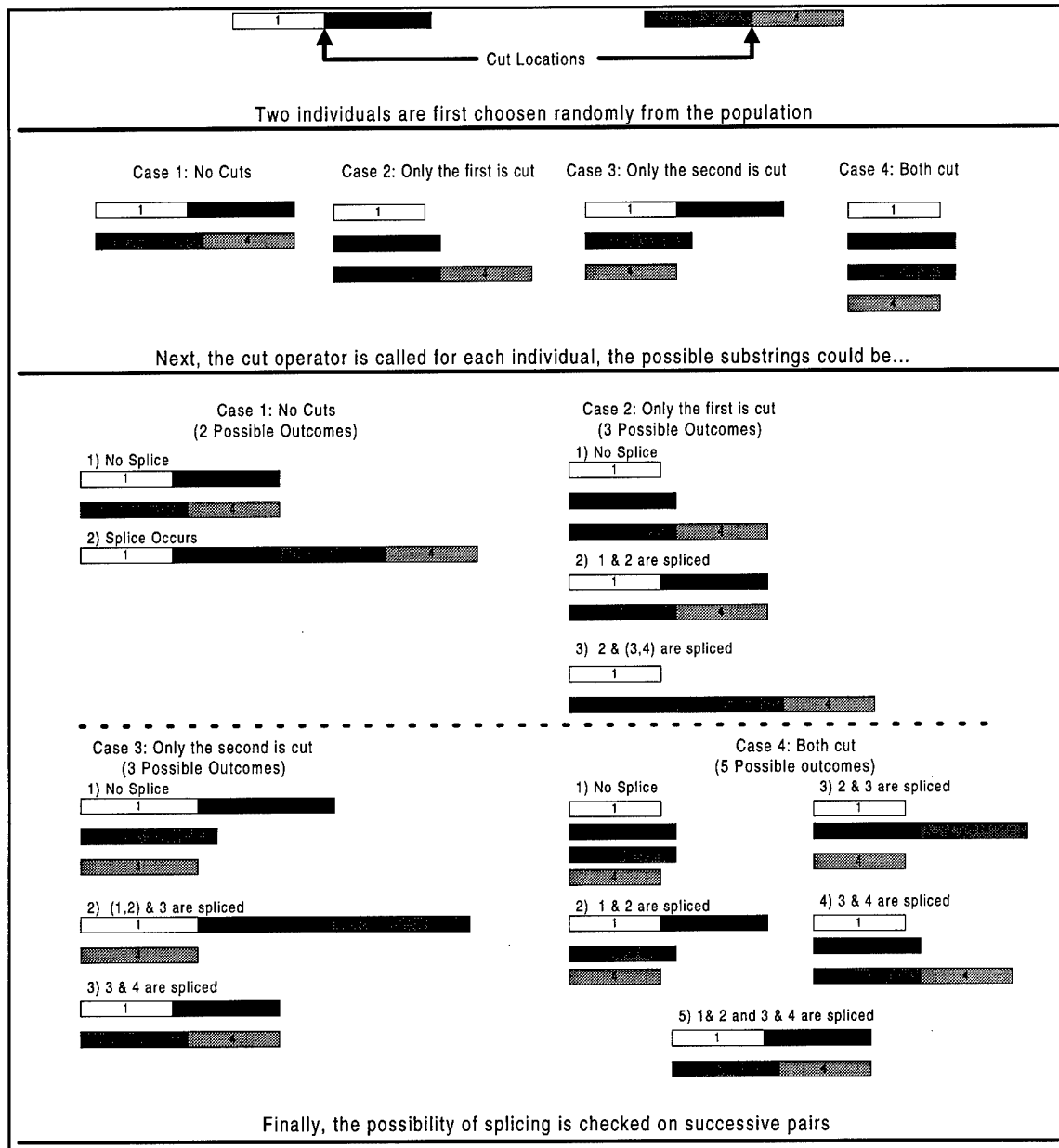


Figure 71: Cut-and-Splice in a Nutshell

E.2 Selfish Gene Genetic Algorithm (SG GA)

The SG GA proposed by Corno, Reorda and Squillero (1998) follows a somewhat nontraditional view of evolution. The SG GA follows a recently proposed view of evolution where the fundamental unit of natural selection is the gene rather than the individual. The selfish gene theory of evolution, proposed by Richard Dawkins in 1976, claims that the individual does not survive, but the genome of the individual is able to

replicate itself into subsequent generations [4]. In the selfish gene concept of evolution, individual genes strive for appearance in the genotype of the individuals, whereas the individual is nothing more than a vehicle allowing the genes to reproduce. Due to the shuffling of genes that takes place during sexual reproduction, "good" genes (i.e., good building blocks) are viewed as those genes that, when combined with other genes, give higher reproduction probabilities to the offspring [4]. Thus, such genes have a higher probability to spread in the gene pool and therefore receive greater representation in future generations.

E.2a SG GA Chromosome Representation: Virtual Gene Pool Growth Rate

The rate at which the size of the virtual gene pool increases is much less than the growth rate found in the mGA (see **Figure 72**). The growth rate of the SG GA is based on the length of the chromosome (i.e. the number of loci) and the allelic alphabet (i.e. the alphabet of values any loci can assume) for each locus. While the allelic alphabet need not be the same for each locus, the graph assumes a binary allelic alphabet for each locus.

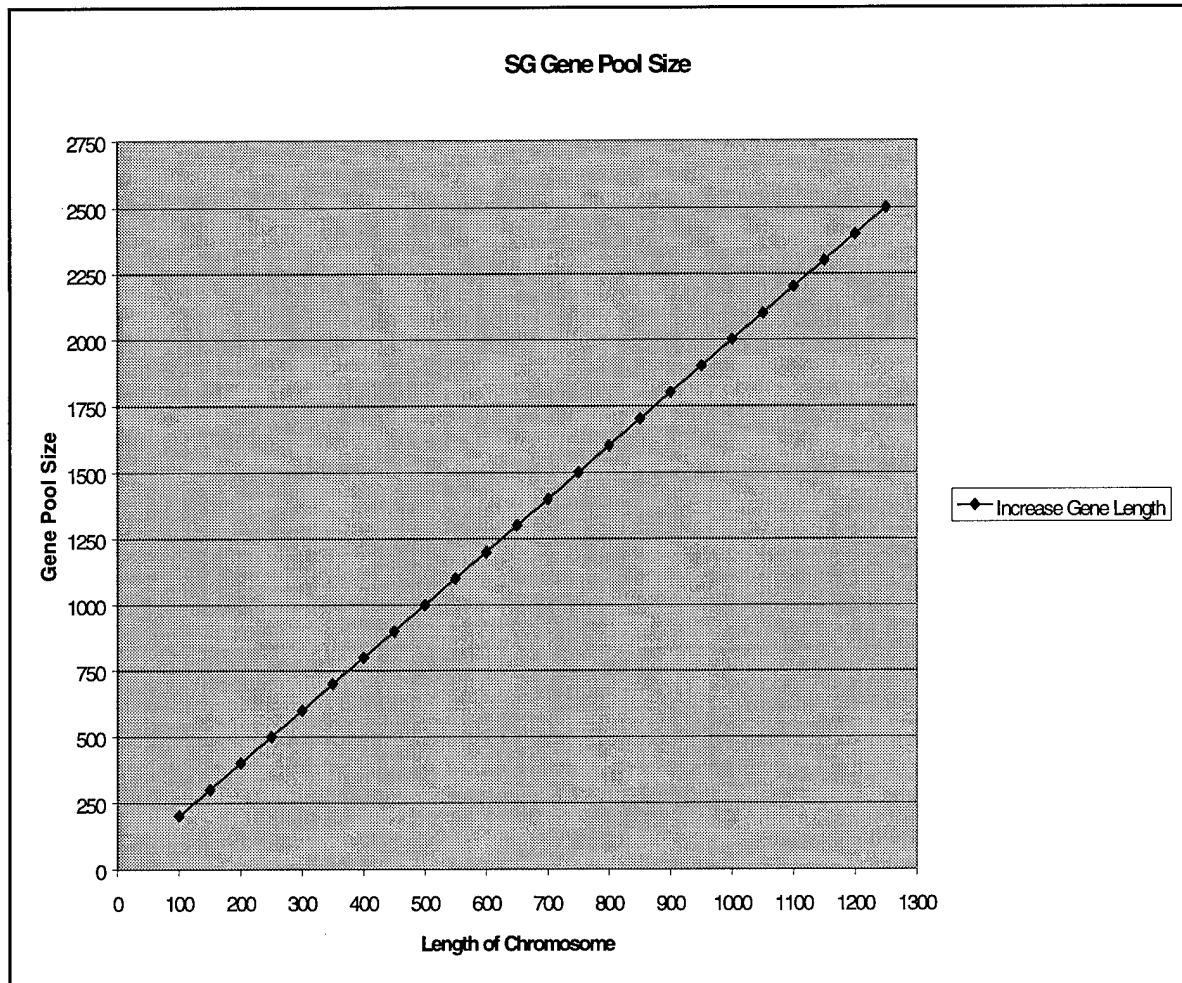


Figure 72: Population Growth Rate for the SG GA

E.2b SG GA Algorithmic Phases: Mutation

The probability of a mutation (P_m) partially controls the rate of convergence of the SG GA. If P_m is set too high, mutants can invade the population and either cause a convergence to a suboptimal point in the landscape or cause enough variation within the allele marginal probability vector to hinder convergence. On the other hand, if P_m is set too low, the alleles converge rapidly to the first optimal solution found by the SG GA.

For example, if we view the search space in only two-dimensional space, the SG GA is traversing a mountain range represented as a line graph. The suboptimal solutions are any locally maximal points that are not the maximum. (See **Figure 73.**) (Reverse this analogy for a minimization problem.)

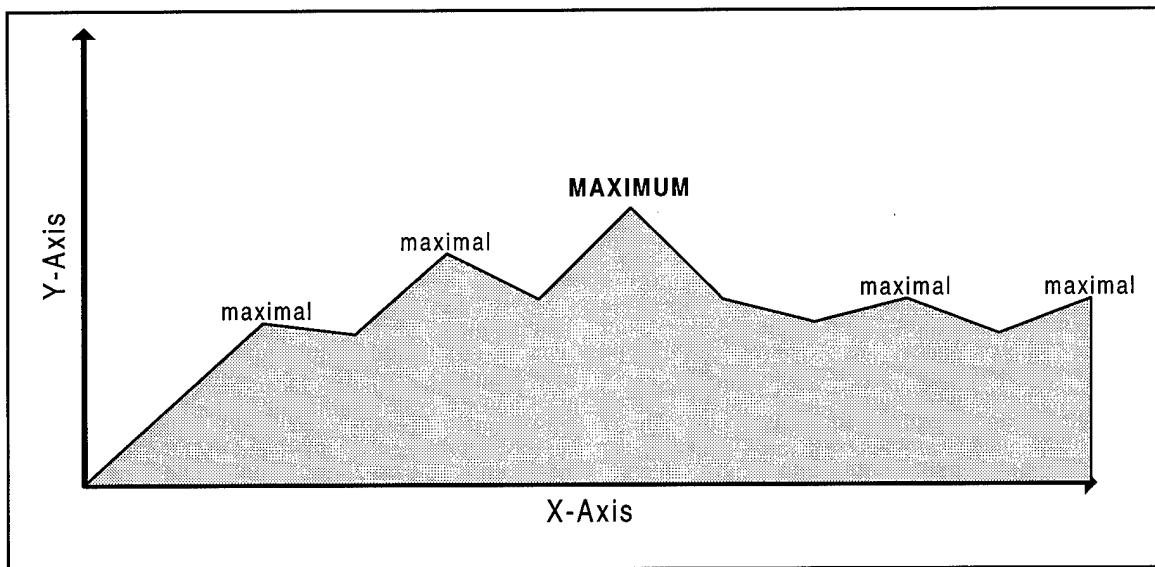


Figure 73: 2D Fitness Landscape

E.2c SG GA Algorithmic Phases: Allele Frequencies and Epsilon (ϵ)

The value of ϵ determines the extent of the positive/negative feedback in the system, and therefore, the balance between a fast convergence towards a local optimum and a broader exploration of the search space [4]. To understand how this feedback is triggered, consider an allele a_{12} ⁷⁵ that produces a better fitness for the individual when allele a_{23} is also expressed. If allele a_{23} increases its frequency, then the individuals with allele a_{12} becomes more likely to win tournaments. This causes allele a_{12} to also increase its frequency. This feedback mechanism quickly drives the virtual population towards a locally optimal solution that includes both allele a_{12} and a_{23} . The convergence speed of the SG can be tuned using this concept [4]. A large ϵ drives the virtual population towards the first local optimal it finds, while a very small value for ϵ makes the VP float for a longer time before converging to on a particular local optimum.

E.3 Gene Expression Messy Genetic Algorithm (GEMGA)

The gene expression messy genetic algorithm (GEMGA) is another compelling investigation into the linkages between genes as proposed by Kargupta in 1996. GEMGA's foundation is rooted in an alternate perspective of blackbox optimization

⁷⁵ Allele representation: $a_{lv} \Rightarrow a = \text{allele}, l = \text{locus}, v = \text{value}$

(BBO) in terms of relations, classes, and partial ordering which Kargupta coins as SEARCH (Search Envisioned As Relation and Class Hierarchizing) [9, 10, 11, 45].

E.3a Chromosome Representation: Gene Representation

Each gene representation in GEMGA contains three values: the *locus*, *allele*, and *weight* [9]. The locus and allele follow the traditional definitions and are used to guarantee positional independence of the gene within the chromosome. The weight, as it corresponds to a gene characteristic, was initially used to explicitly evaluate the relation space [9], but this gene characteristic has evolved. In 1997, Kargupta changed the weight's role to that of modeling the class space [45]. Then, in the most current version of GEMGA (1998), the *weight characteristic of a gene* has changed names and roles [10]. It is now called a *capacity*, and it represents the possibility of change for that particular gene. Therefore, it no longer models the class space. This gene characteristic still requires a positive real value lower bounded by zero [9, 11, and 45]. The capacity of a gene is determined by the transcription operator, and will be further explained in **Section 3.4.2**.

E.3b Algorithmic Phases: Initialization

We can estimate the size of the population required to investigate order- k relationships between genes by varying c in order to model different fitness variances among the initial population. Based upon **Equation 18**, **Figure 74** estimates the size of the initial population required if we are looking for 3rd-order linkages with varying c -value and increasing n until the 3rd-order problem is solvable. The increasing c -value indicates more fitness variance in the initial population. As with previous graphs, **Figure 74** assumes a binary allelic alphabet. **Figure 74** indicates that as the c -value increases the size of the population must increase by a factor of eight, but since the c -value represents the fitness variation amongst the population members there is no *a priori* method of determining the initial population size. Therefore, if our search landscape contains extremely high fitness values (1×10^{100}) and extremely low fitness values (-1×10^{100}), GEMGA may be unable to find the optimum without enumerating most of the landscape in the initial population.

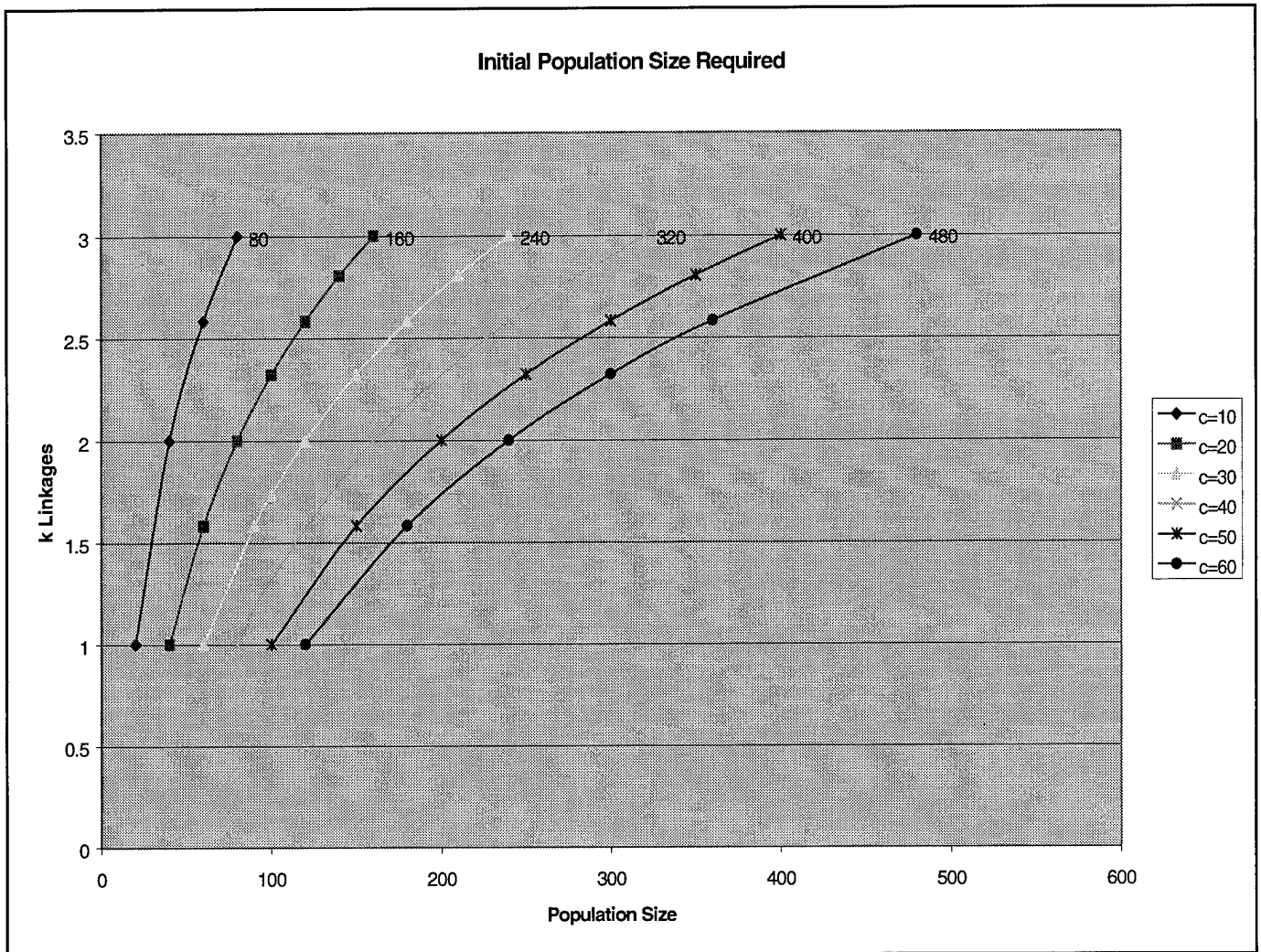


Figure 74: GEMGA Population Growth vs. Increasing Initial Variance

E.3c Algorithmic Phases: RecombinationExpression

The RecombinationExpression phase of GEMGA begins with the “modified” population. The RecombinationExpression stage is actually two separate subphases: PreRecombinationExpression and RecombinationExpression. The RecombinationExpression phase repeatedly applies these two subphase until some predefined stopping condition is met.

The first subphase is the application of the PreRecombinationExpression operator. During this subphase, the PreRecombinationExpression operator is applied to the population to determine the clusters of genes precisely defining the relations among

those instances of genes considered [10]. For example, two chromosomes are randomly selected from the population. One is arbitrarily designated as dominant⁷⁶. The linkage set of the dominant individual is compared to the genes from the other chromosome. If the dominant chromosome linkage set members are contained within the other chromosome and they have the same value and capacity, then they are grouped and extracted as a set. In the linkage set of the dominant chromosome, each gene has its weight incremented by a predefined amount (i.e. an algorithm parameter) [10]. If this new linkage set is not already included as a linkage set of the recessive chromosome, it is included as a new linkage set and the different factors are initialized as discussed in the transcription operator.

After a prespecified number of trials (i.e. another algorithm parameter), a $\ell \times \ell$ *conditional probability matrix* is formed. The matrix entries indicate the probability of linkage between two genes [10]. Finally, the maximum value for each row of the matrix is computed. The genes within some predetermined ϵ of the maximum are retained as the linkage set for each gene row, and the weight is set to the average value of the entire matrix. This operator is only applied in the first generation and the linkages are never recalculated [10].

In the second phase of the RecombinationExpression stage, the GEMGA Recombination operator is applied. This operator implements crossover and reproduction/selection in GEMGA. First two random chromosomes are selected from the population, and each is copied (i.e. A to A', B to B'). One of the selected chromosomes is chosen as the donor of genetic material (A'). An element of the A' linkage set is transferred to B' based on a linearly combined factor of its weight and goodness. The transfer of corresponding genes between the two chromosomes (A' and B') is based on whether or not the goodness values of the disrupted linkage set for B' are less than that for A'.

Once the linkage sets of the two offspring are adjusted, they undergo a fitness evaluation. Furthermore, depending on whether the fitness of B' is decreased or not, the goodness of the selected linkage set from A' is either increased or decreased [10]. The product of this operator is four unique chromosomes (e.g., 2 parents and 2

⁷⁶ The dominant chromosome is arbitrarily chosen from the two chromosomes.

children). Based on the fitness of the four individuals, two chromosomes with the best fitnesses are returned to the population.

E.4 Linkage Learning Genetic Algorithm (LLGA)

The LLGA was first proposed as a new linkage investigating algorithm by Harik in 1996. Harik argues that other implementations of genetic algorithms do not take explicit advantage of “tight linkages” early enough in their algorithmic processing. If they did (as does the LLGA), then they would be able to solve “difficult problems [7].” The LLGA takes advantage of tight linkages between genes by using a new two-point crossover operator and a different chromosome representation.

E.4a LLGA Algorithmic Phases: Exchange Operator

The follow figure illustrates the LLGA’s exchange operator.

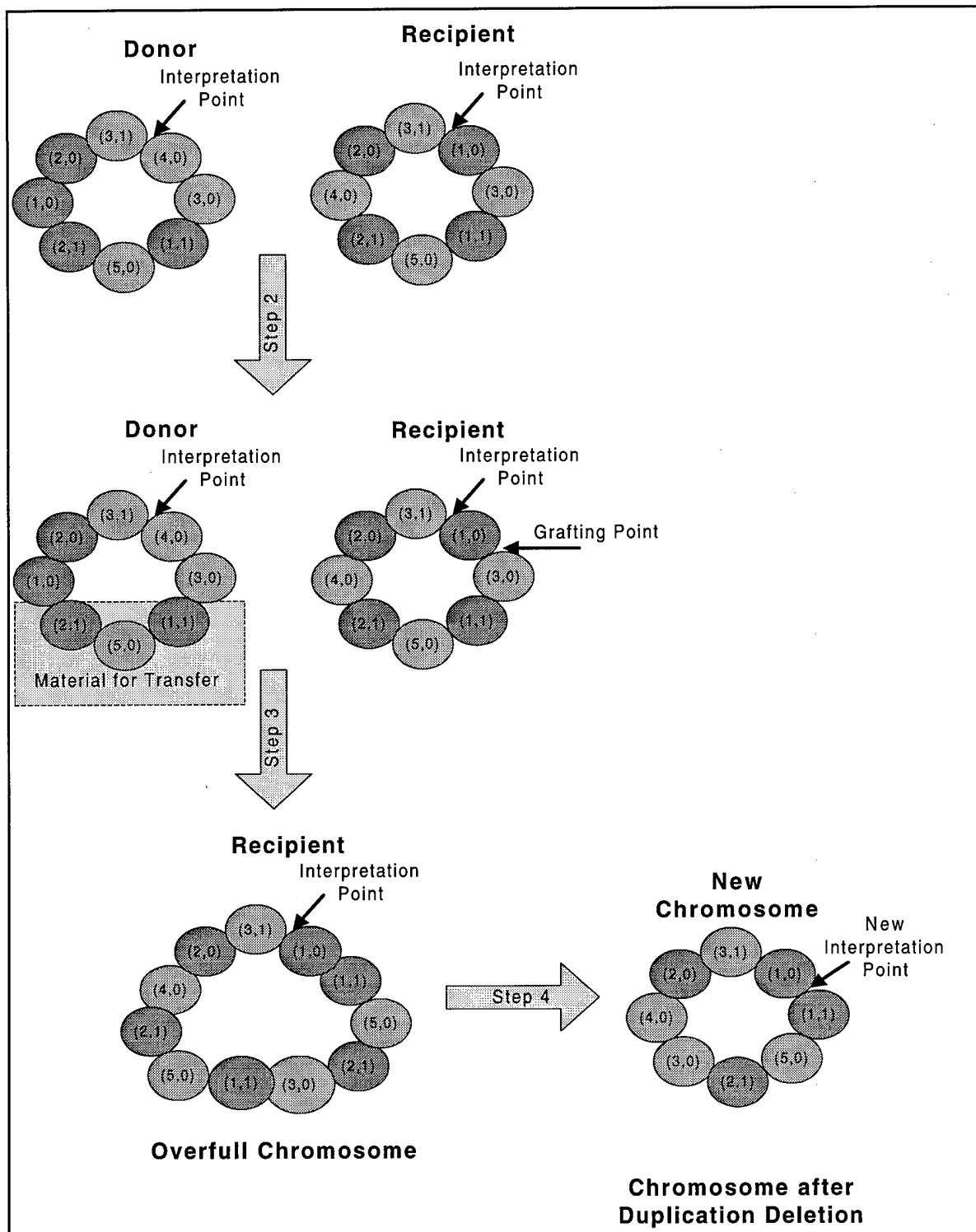


Figure 75: An Overview of the Exchange Operator

E.4b LLGA Algorithmic Phases: Preconvergence Avoidance and Introns

In order to avoid preconvergence, the LLGA requires an “exponentially larger number of introns encoded” into the chromosome to facilitate the Schema Theorem [7, 17]. Basically, the LLGA is trying to force linkages within building blocks to become shorter in the classical sense while counter-acting the disruptive affects of crossover. Historically, a mutation operator battles the GA’s tendency to preconverge by “reseeding” the population by mutating genes within some number of chromosomes. It is implicitly assumed that the mutated chromosomes represent previously unseen terrain of the search space. This enables the GA to escape local minima. Since a single chromosome in the LLGA population represents the complete search space, mutation, in the classical sense, would not make sense⁷⁷. Therefore, the number of introns coded into the chromosome *a priori* plays a major role in assuring that the LLGA adequately searches the landscape before converging to a particular minima. The number of introns required in the chromosome is a function of how disruptive the crossover operator is encouraged to be, as well as the number of exons. **Equation 43** mathematically represent this notation expressed as a probability (P) that crossover is disruptive.

$$P = \frac{\text{number of exons}}{(\text{number of exons}) + (\text{number of introns})}$$

Equation 43: Introns Required per X Exons

Typically, the smaller the value of P the more likely it is that building blocks are preserved. Crossover always is disruptive when $P = 1$.

This occurs when there are no introns. If we relate **Equation 43** to a chromosome which has 240 exons, and we desire crossover to be disruptive only 0.01% of the time, then,

⁷⁷ The classical mutation operator would actually remove alleles from the population instead of reintroducing them. On the other hand, a LLGA mutation operator could be constructed, but instead of mutating allelic values, it should mutate the interpretation point. This would allow previously unexpressed genes to surface.

$$0.01 = \frac{240}{240 + (\text{number of introns})}$$

$$0.01(240 + (\text{number of introns})) = 240$$

$$2.4 + 0.01(\text{number of introns}) = 240$$

$$\text{number of introns} = 23,760$$

Figure 76 indicates how quickly the number of introns must grow in order to counteract the effects of disruption.

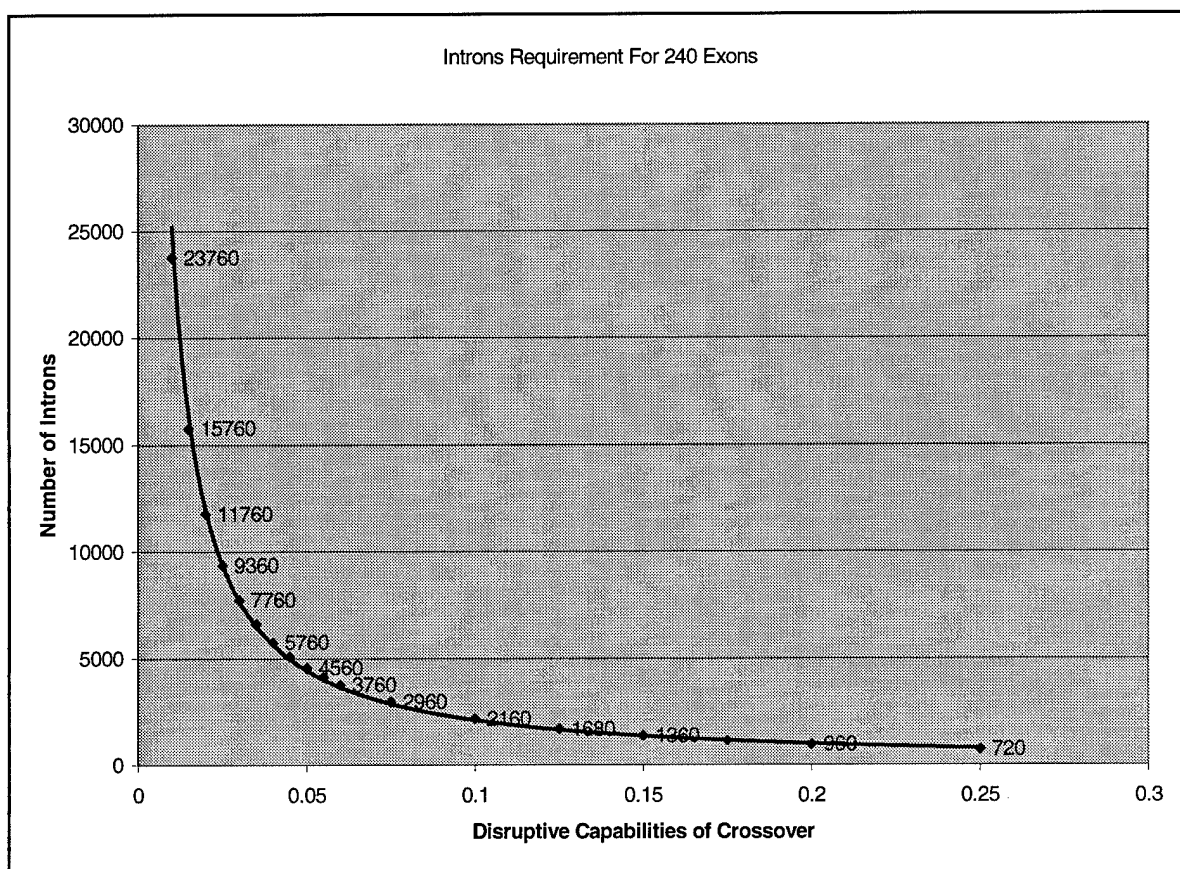


Figure 76: Intron Requirement Trend

Appendix F: Software Locations

F.1 Source Code

The source code corresponding to these implementations is found on the AFIT Parallel and Distributed Lab Network-of-Workstations (NOW) room 243 building 640 under the following directory:

`~genetic/Software/`

This directory is composed of the following subdirectories and a short description of their contents is provided:

- ♦ Linkage_Learning
 - ♦ **/original**: contains the original untampered Linkage Learning genetic algorithm (LLGA) source code.
 - ♦ **/sllga**: contains the modified LLGA-Protein Structure Prediction (PSP) source code.
 - ♦ **/pllga**: contains the modified parallel LLGA- PSP source code.
 - ♦ **/constrained_charmm**: contains the modified CHARMM source code.

F.2 Input and Output Files

Gualke developed a good description of the required input files for the PSP calculations and the basic set of output files AFIT GAs produce [73]. The reader is referred to his thesis for their coverage. **NOTE: THE RTF (MOLECULE TOPOLOGY FILE) FILE AND THE PARM.PRM (PARAMETER FILE) NEED TO COME FROM THE SAME QUANTA VERSION. IF THE DON'T, THE AFIT TOOLBOX IMPLEMENTATION WILL EITHER HANG OR RETURN ERRONEOUS RESULTS.** The LLGA requires a single input file in the following form:

```

*****
# inputfile1: a sample input file for the LLGA.
# This is the 19 exponentially scaled BB problem that is described
# in Georges Harik's thesis (pp 89-90).
# Each BB is a 4-bit trap function.
# The signal is 0.4 (global is 1, deceptive is 0.6).
# The BBs are scaled by a factor of 7.
#
#building_blocks
# 1-19 : low & high BB - means all BB will be tested using "trap"
# trap : objective function
# 4 : BB size
# 0.4 : signal
# 7 : weight
#coding_genes
# each gene is made up of 4 BBs
#noncoding_genes
#
#selection_operator
# tournament_with_replacement doesn't work
# tournament_without_replacement doesn't work
# tournament_between_generations
#pcross
# probability of crossover
#stop_criteria
#
#seed
# random number generator seed
#report_population
# turned off
#report_best_individual
# turned off
#BBtemplate_filename
# this file should contain a string from 0 - coding_genes for
# a BB template. It should be in the canonical form.
*****
BEGIN
building_blocks:      24
                    1-24 CHARMM_EVAL 10 0.7 0
coding_genes:        240
noncoding_genes:     4560
popsize              4
selection_operator:   tournament_between_generations
selection_rate:       4
pcross:              0.70
stop_criteria:        gen= 5000
seed:                0.37234535
report_population:    off
report_best_individual: off
BBtemplate_filename:  best.txt
END

```

Figure 77: Sample LLGA Input File

The LLGA implementations creates output files: timing.txt, charmm_molecules.txt, and output.txt. timing.txt contains the timing characteristics for each LLGA execution broken

down as the total program execution and the CHARMM energy model execution. charmm_molecules.txt contains every chromosome sent to the fitness function as well as the corresponding fitness values. Finally, output.txt contains population parameters for the complete LLGA execution. Be careful! This file grows rapidly. A typical 5,000 generation run of the LLGA results in 100Mb of information stored in output.txt. In order to record these files, the charmm *define* in output.h needs to be specified and the source code recompiled for charmm_molecules.txt and the output *define* needs to be specified for output.h.

This appendix documents the structural design of the AFIT implemented CHARMM energy model. First a top-level structure chart is given followed by the top level CHARMM structure chart (func). Finally, the structure charts for the local minimization techniques are include.

```

graph TD
    Root["charmm_eval  
{energy.c}"]
    
    Root --> Frpmn["frpmn  
{frprmn.c}"]
    Root --> Func["func  
{energy.c}"]
    Root --> DataTypes["double *  
Replacement  
Indep_dihedral  
TwoPl  
Minimization  
max_range  
num_dihedral  
Slice  
int  
char []  
double"]
    
    Frpmn --> Frpmn_Int["int"]
    Frpmn --> Frpmn_Double["double"]
    Frpmn --> Frpmn_DoubleArray["double []"]
    Frpmn --> Frpmn_FuncPtrs["Function Ptrs {func}"]
    Frpmn --> Frpmn_FuncPtrsArray["Function Ptrs {func}"]
    
    Func --> Func_DependAngle["Depend_angle"]
    Func --> Func_DependBond["Depend_bond"]
    Func --> Func_IndepDihedral["Indep_dihedral"]
    Func --> Func_NonBond["non_bond"]
    Func --> Func_FuncCount["func_count"]
    Func --> Func_FixedEnergy["fixed_energy"]
    Func --> Func_IndepAngle["Indep_angle"]
    Func --> Func_OneFour["One_Four"]
    Func --> Func_Atom["atom"]
    Func --> Func_DepDihedral["dep_dihedral"]
    
    DataTypes --> DataTypes_DoubleStar["double *"]
    DataTypes --> DataTypes_Replacement["Replacement"]
    DataTypes --> DataTypes_IndepDihedral["Indep_dihedral"]
    DataTypes --> DataTypes_TwoPl["TwoPl"]
    DataTypes --> DataTypes_Minimization["Minimization"]
    DataTypes --> DataTypes_MaxRange["max_range"]
    DataTypes --> DataTypes_NumDihedral["num_dihedral"]
    DataTypes --> DataTypes_Slice["Slice"]
    DataTypes --> DataTypes_Int["int"]
    DataTypes --> DataTypes_CharArray["char []"]
    DataTypes --> DataTypes_Double["double"]
  
```

175





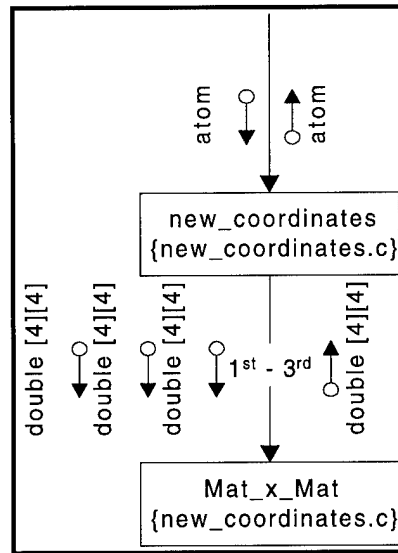


Figure 81: New Coordinates Module Expansion

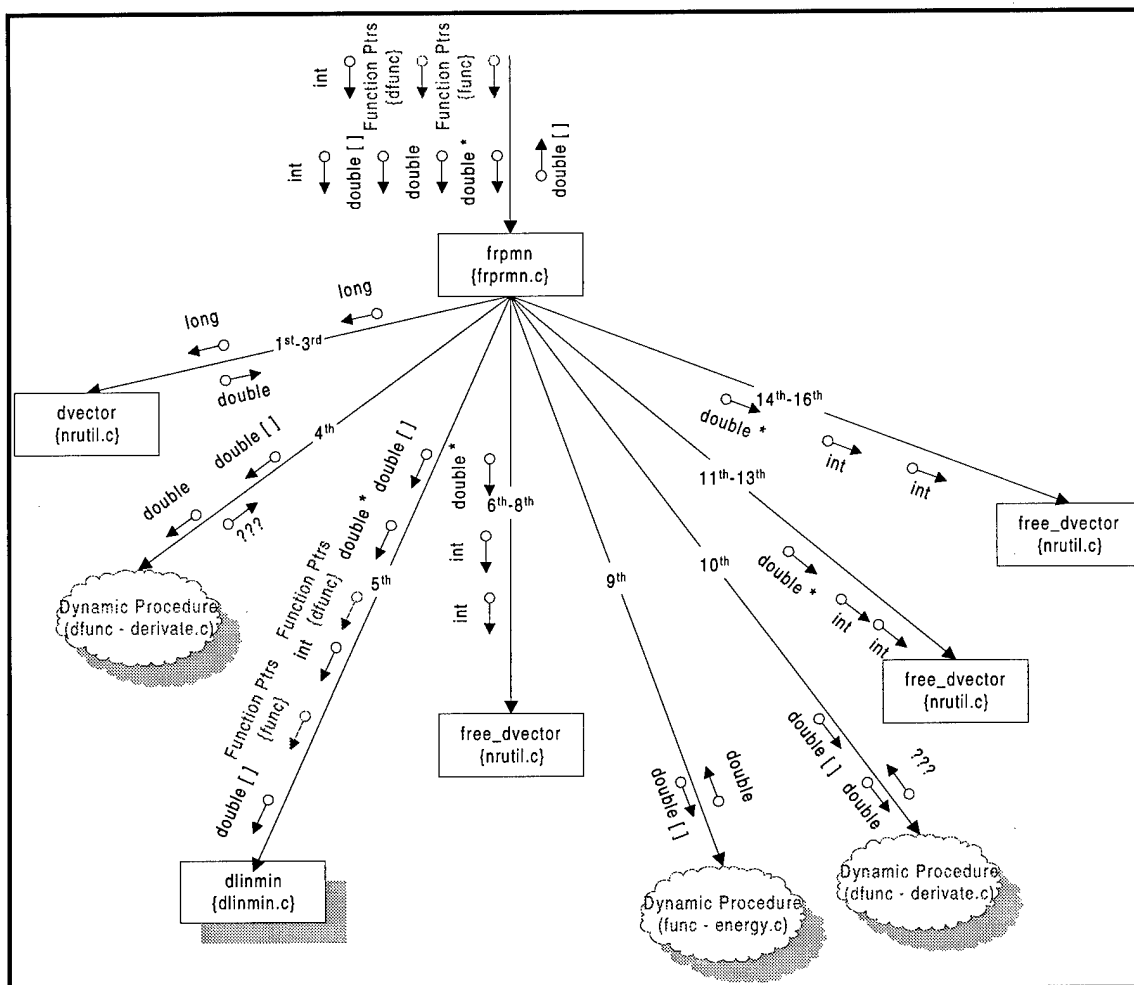


Figure 82: Local Minimization Top-Level Data and Structure Diagram

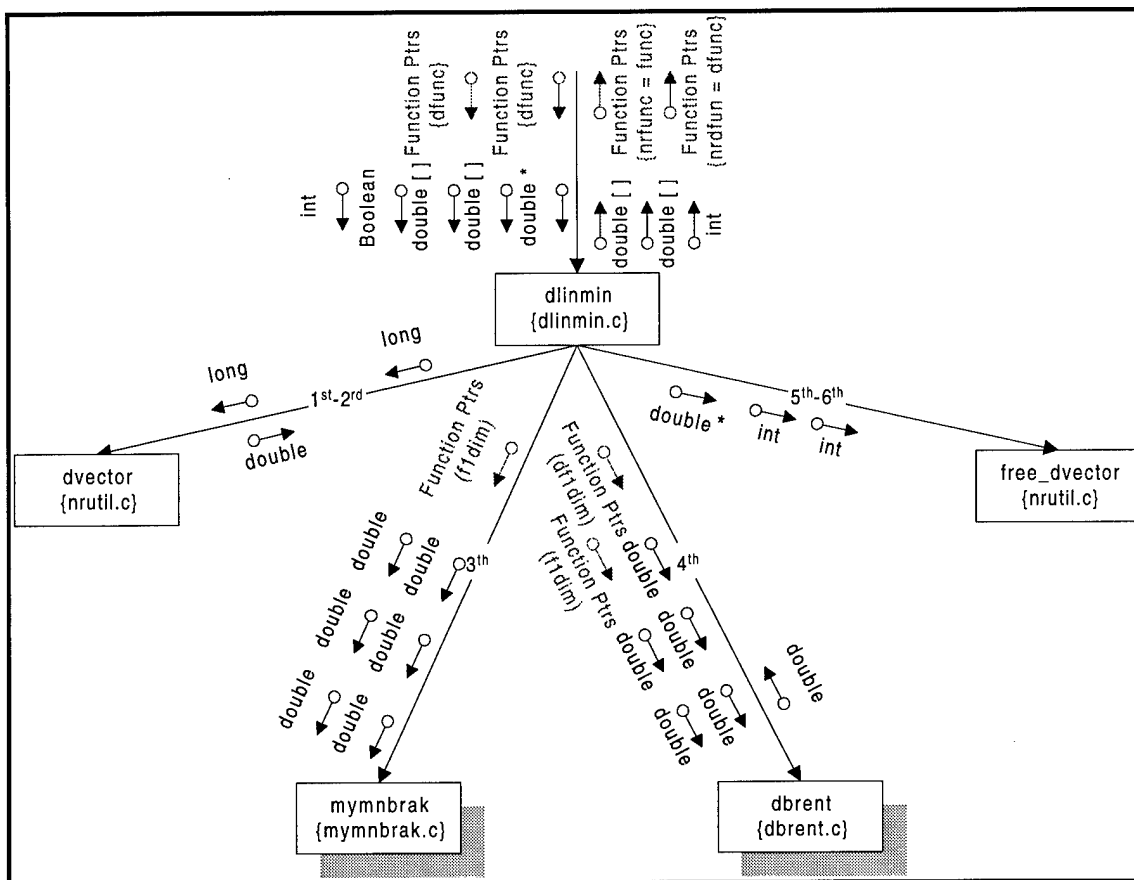


Figure 83: Module dlinmin Expansion

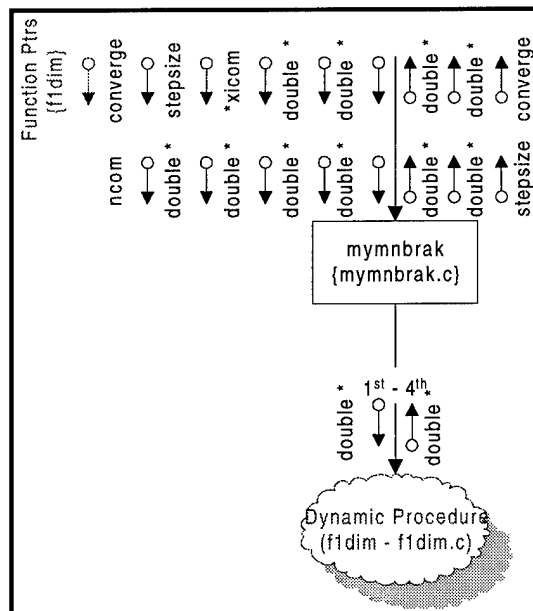


Figure 84: Module mymnbrak Expansion

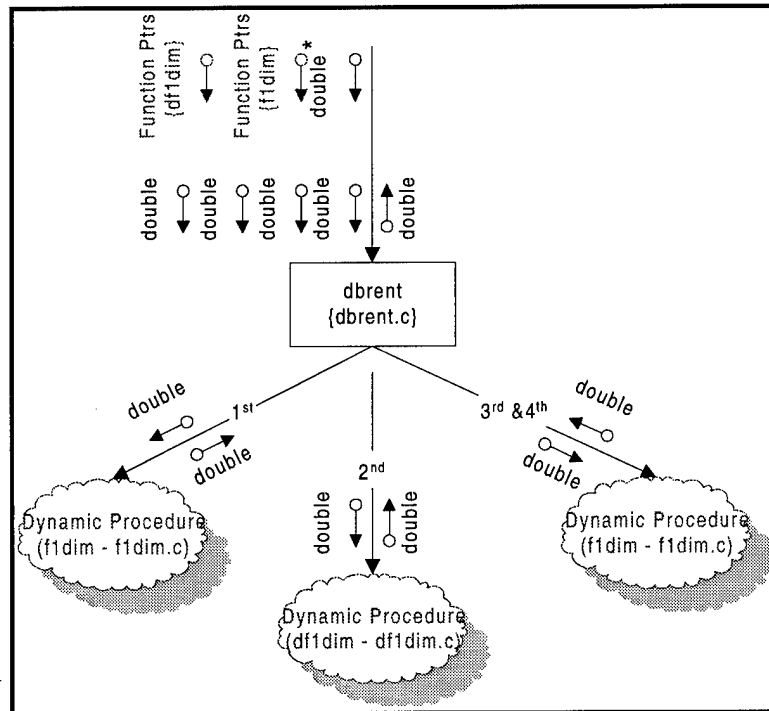


Figure 85: Module dbrent Expansion

Appendix H: Object Classes for the Redesigned LLGA

The overall class decomposition for the redesigned LLGA is in **Chapter 4 Figure 27**. This appendix lists the corresponding Rumbaugh diagrams for each object class. The text in red indicates parts of Harik's hierarchy that we changed.

llga {llga.hpp}
Attribute n_BB: static int; { number of building blocks and subfunctions } *subfunc: static Subfunc; {types of subfunction: data type defined here} *gene_Subfunc: static int; {data structure that given a gene finds the BB that the gene belongs to.} seed: static double; {seed for random number generator} coding_genes: static int; {problem length - equivalent to sGA chromosome length - exons} noncoding_genes: static int; {number of introns} total_genes: static int; {intron + exons} popsize: static int; {population size} pcross: static double; {probability of crossover} selection: static PSO; {selection method - data type defined here} s: static int; {selection rate - tournament size} stop_criteria: static PSC; {stop criteria - data type defined here} stop_criteria_arg: static int; {stop criteria argument (ex: maxgen)} gen: static int; {generation counter } Initialization_Accomplished: static bool; {flag for charmm initialization } num_processors: static int; {number of processors used in parallel } myid: static int; {my processor identification } BBtemplate_filename[20]: static char; {filename of BB template file}
Operation: {this is the main loop for the program}

report {llga.hpp}
Attribute population: static bool; {flag - on = report population / off = don't} best_individual: static bool {flag - on report best individual / off = don't}
Operation: {this class is used as flags to tell what to report}

timing {llga.hpp}
Attribute llga_start_time: static double; {start time for complete run} llga_stop_time: static double; {stop time for complete run} charmm_start_time: static double; {start time for charmm eval} charmm_stop_time: static double; {stop time for charmm eval} charmm_ave_time: static double; {average time for charmm evals} times_charmm_called: static int; {number of time charmm called} timeF: static ofstream; {output filename}
Operation: {this class is used as flags to tell what to report}

llga_io {llga_io.hpp}
Attribute:
Operation: void read_parameters (ifstream &in); void report (int gen, population *pop, ofstream &reportF, ofstream &convergenceF, ofstream &maxLinkageF, ofstream &avgLinkageF); void print_header (ofstream &out); void help(); int find_testfunc (char *key); int find_selection_op (char *key); int find_sc (char *key);

util {util.hpp}
Attribute:
Operation: void makeshuffle (int *shufflearray, int n); void errorcheck (char *str, bool condition); bool is_odd (int x); bool is_even (int x); int Min (int a, int b); int Min3(int a, int b, int c); double sqr (double x);

random.h
Attribute:
Operation: void randomize (double seed); int flip (double probability); int rnd (int low, int high); double random01 ();

population {population.hpp}
Attribute: *Chromosome: chromosome; {array of chromosomes} *MatingPool: int {mating pool} *BBtemplate: chromosome; {a chromosomes} *Worst: chromosome; {the worst chromosomes in the population} *BB: int; {number of building blocks across the pop} MaxFit: double; {max fitness} MinFit: double; {min fitness} AvgFit: double; {average fitness} MaxBBs: double; {max # of BBs in a singel individual} AvgBBs: double; {avg # of BBs per individual} MaxChromLength: int; {max chromosome length} MinChromLength: int; {min chromosome length} AvgChromLength: double; {avg chromosome length} Best: int; {index of best individual} *MaxLinkage: double; {max linkage of BBs} *AvgLinkage: double; {avg linkage of BBs}
Operation: double compute_maxobj(); int count_copies_of_BB(int BB_number); void initial_gen_random(); int tournament_winner(int *shuffle, int &pick, int s); population(); population(population &pop); ~population(); void initial_generation (fstream &bbF); population * selection(population *children); void recombination(population *children); void statistics(); void save_worst(); void evaluate(); double maxChromLength(); double minChromLength(); double avgChromLength(); double maxfit(); double minfit(); double avgfit(); double maxBBs(); double avgBBs(); int best(); int copies_of_BB(int i); chromosome & population[](int index); population & operator=(population & pop); ostream &operator void print(ostream &out); void print_BB(ostream &out); void print_worst(ostream &out); friend void tselect_without_replacement(population &pop); friend void tselect_with_replacement(population &pop); friend void tselect_between_generations(population &pop, population &children); friend int sc_maxgen(const population &pop, int gen); friend int sc_all_or_none_BB(const population &pop, int dummy);

chromosome (chromosome.hpp)
Attribute: Genes: geneArray; {array of introns and exons} Size: int; {size of chromosome} Fitness: double; {fitness value} Canonical: geneArray; {array of expressed introns} *Linkage: double; {max linkage of BB}
Operation: chromosome(); chromosome(chromosome&pop); ~chromosome(); chromosome & chromosome::operator = (chromosome &c); void random (); void insert(gene &g); void compute_linkages(); void display(ostream &out); void asGeneArray(geneArray &temp, int &tempsize); void express(geneArray &temp, int &tempsize); void express(); void evaluate(); void inject(); void display_BB(ostream &out); void Initialize_BB(ifstream &bbF); bool hasBB(); int get_random_point(); int size(); double fitness(); int correctBBs();

geneArray geneArray.hpp
Attribute: Size: int *Genes: gene
Operation: geneArray(int size); geneArray(geneArray &v); ~geneArray(); gene & opertor[] (int i); ostream &opertor; int size();

gene (gene.hpp)
Attribute: Locus: int Allele: char
Operation: gene(int p, char a); gene(gene &g); ~gene(); void set(int p, char a); void set_allele(char a); void set_allele(int a); void set_locus(int p); void random(); void flip(); char allele(); int locus(); bool is_intron(); gene &operator=(gene &g); ostream &operator;

objfunc (objfunc.hpp)
Attribute:
Operation: double objfunc (geneArray &v) double trap (int lbits, double signal, int *locus, geneArray &x) double ttmp (int lbits, double signal, int *locus, geneArray &x) double CHARMM_EVAL (int dummy1, double dummy2, int *dummy3, geneArray &x)

Appendix I: Newman Projection for the Each Dihedral Constraint

This appendix contains Newman Projections to aid the understanding of the constraints placed on each dihedral angle. View the center of the circle as the first atom in the atom pair. Then, the indicated regions depict the possible location of the corresponding next atom.

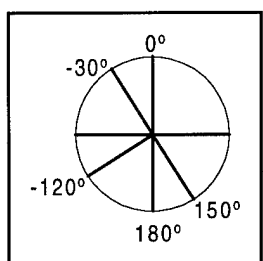


Figure 86: Phi Constraints

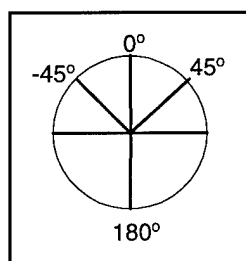


Figure 87: Glycine Phi Constraints

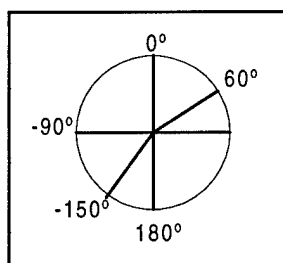


Figure 88: Psi Constraints

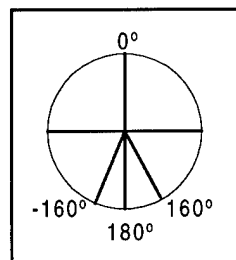


Figure 89: Omega Constraints

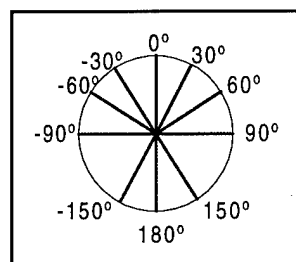


Figure 90: Chi Constraints

Appendix J: Statistics Explained

No experiment is complete without a thorough and complete analysis of the collected data. This section covers the types of analysis that could be conducted for each set of experiments including analysis of variance and the Kruskal-Wallis H test. We shall use ANOVA F-test in conjunction with Kruskal-Wallis H Test to show that valid comparisons can be made between separate GA populations.

J.1 Analysis of Variance Testing (ANOVA)

“Essentially, this analysis determines whether the discrepancies between the *treatment* averages are greater than could be reasonably be expected from the variation that occurs within the treatment classification [76].” Historically, statistical publications use the term “treatments” to refer to different populations [74]. ANOVA can be used to compare a number of population means simultaneously. Thus, the need to make a large number of two-sample tests is avoided [74]. The assumption made about the distribution of the data becomes important when we want to use ANOVA to make decisions. For instance, when we assume the observations come from a normal distribution whose variance does not depend on treatment levels, then the summary statistics (MS_E , $MS_{\text{treatment}}$, F) have known distributions. It allows us to answer “if there is no treatment effect, is it likely we would see an F-statistic this large?” If the answer is “the probability is too small (i.e., a small p-value), then we conclude that there is a treatment effect.” The central limit theorem provides the basis for the explanation of the observed fact that many random variables tend to be normally distributed. Referring to **Table 19**, the population for each experiment is only 50 members out of a search space of 2^{240} possibilities. Therefore, we cannot implicitly assume that our population is normally distributed and compare two individuals from separate population (i.e. comparison of two unique optimal solutions). Hence, the supporting Kruskal-Wallis H Test in these case. If both tests agree that the individuals can be compared, then valid conclusions can be made between separate test.

On the other hand, we can use the F-test when we are comparing the average energy obtained by two unique algorithm executions because we are taking the average over the entire population (fifty members). The Central Limit Theorem indicates that if we take a large number of observations independently from the treatment groups and take twice their average, the average will behave as if it came from a normal distribution.

The upshot of this is that if you construct a ANOVA table and calculate the F-test, this will then have an approximate F distribution.. Furthermore, the population size inside the GA has little to do with the distribution of the output (i.e., the average energy).

Table 46 is an example of a two-way ANOVA table for the test identified in **Section 5.3**.

Random Number	Selection Scheme	
	Experiment 1 (LLGA)	Experiment 1 (pLLGA)
0.014150	Execution 1	Execution 1
⋮	⋮	⋮
0.928304	Execution 7	Execution 7

Table 46: Example ANOVA Table

For hypothesis testing, the model errors are assumed to be independent normally distributed random variables with mean of zero and variance of σ^2 . The basic two-way ANOVA test follows the mathematical model [76]:

$$y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \epsilon_{ijk} \quad \begin{cases} i = 1, 2, \dots, a \\ j = 1, 2, \dots, b \\ k = 1, 2, \dots, n \end{cases}$$

where, μ is the overall mean effect, τ_i is the effect of the i th level of the first factor A, β_j is the effect of the j th level of the second factor B, $(\tau\beta)_{ij}$ is the interaction between τ_i and β_j , and ϵ_{ijk} is the random error component.

Equation 44: Two-Way ANOVA Design

Usually, ANOVA analysis is presented in a table similar to **Table 47**, which shows the general decomposition of a two-way ANOVA analysis.

Source(s) of Variation	Sum of Squares (SS)	Degrees of Freedom (DoF)	Mean Square (MS)	F _o
Treatment A	SS _A	a - 1	SS/DoF	MS _A / MS _E
Treatment B	SS _B	b - 1	SS/DoF	MS _B / MS _E
Interaction	SS _{AB}	(a - 1)(b - 1)	SS/DoF	MS _{AB} / MS _E
Error	SS _E	ab(n - 1)	SS/DoF	
Total	SS _T	abn - 1		

Table 47: Two-Way ANOVA Decomposition Table

The terms of the table are computed in the following manner [77]:

$$SS_A = bn \sum_{i=1}^a (\bar{Y}_{i..} - \bar{Y}_{...})^2 \text{ where } \bar{Y}_{i..} = \frac{\sum_{j=1}^b \sum_{k=1}^n Y_{ijk}}{bn}$$

$$\bar{Y}_{...} = \frac{\sum_{i=1}^a \sum_{j=1}^b \sum_{k=1}^n (\bar{Y}_{ijk})}{abn}$$

where,

$$SS_{AB} = n \sum_{i=1}^a \sum_{j=1}^b (\bar{Y}_{ij.} - \bar{Y}_{i..} - \bar{Y}_{.j.} + \bar{Y}_{...})^2 \text{ where } \bar{Y}_{ij.} = \frac{\sum_{k=1}^n Y_{ijk}}{n}$$

$$SS_B = an \sum_{j=1}^b (\bar{Y}_{.j.} - \bar{Y}_{...})^2 \text{ where } \bar{Y}_{.j.} = \frac{\sum_{i=1}^a \sum_{k=1}^n Y_{ijk}}{an}$$

and

$$SS_E = n \sum_{i=1}^a \sum_{j=1}^b \sum_{k=1}^n (\bar{Y}_{ijk} - \bar{Y}_{ij.})^2$$

Finally,

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E$$

J.2 Kruskal-Wallis H Test

The Kruskal-Wallis test is used in place of the ANOVA test when the treatments cannot be assumed normally distributed. This is the case when a GA uses a small treatment populations, and therefore, the central limit theorem does not apply. This test is used to validate ANOVA results because of the small treatment populations.

The Kruskal-Wallis test uses a ranking method [74]. Let n_i be the number of observations in the i^{th} sample. k samples are grouped together and ranked from smallest to largest, substituting the appropriate rank for 1 to n_i observations. (Observations with the same values are given the average of their ranks.) The sum of the ranks for each sample is then computed:

$$H = \frac{12}{n(n+1)} \sum_{i=1}^k \frac{R_i^2}{n_i} - 3(n+1)$$

where, R_i is the sum of the ranks from the i^{th} sample

Equation 45: Kruskal-Wallis H Test

If each sample consists of at least five (5) observations, then H can be approximated by a chi-square distribution with $k - 1$ degrees of freedom [74]. If five observations can not be made, then the exact distribution of H can be found and critical values derived. If $H \leq \chi^2$, then we can accept the null hypothesis. On the other hand, when $H > \chi^2$ we reject the null hypothesis.

J.3 The Central Limit Theorem

The underlying principle which allows the application of ANOVA testing is the Central Limit Theorem [Theorem 1]. We can apply this theorem when enough samples are drawn from the populations. As a rule of thumb, the Central Limit Theorem can be applied to an experiment where at least 30 samples are used. The central limit theorem provides the basis for the explanation of the observed phenomenon that many random variables tend to be normally distributed.

Central Limit Theorem : Let X_1, X_2, \dots be independent, identically distributed random variables, each having mean and standard deviation $\sigma > 0$. Let $S_n = X_1 + \dots + X_n$. then for each $x < y$,

$$\lim_{n \rightarrow \infty} P \left[x \leq \frac{S_n - n\mu}{\sigma\sqrt{n}} \leq y \right] = \Phi(y) - \Phi(x),$$

where Φ is standard normal distribution function.

Theorem 1: Central Limit Theorem

Appendix K. Material for Test Platforms

K.1 AFIT Network Of Workstations (NOW)

The AFIT NOW consists of six (6) Sun workstations connected via a high-speed Myrinet switch⁷⁸ and a 10 Mbit ethernet hub [82]. The AFIT NOW has been in existence in its current configuration since October 1996.

The workstations comprising the NOW are Sun Ultra Sparc Models 170 and 200. The heart of the Ultra is a 170 or 200 MHz, four-way superscalar Sparc version 9 processor. There are two integer arithmetic logic units (ALUs) and 2 floating point (FP) ALUs. The FP ALUs are pipelined with an FP add or multiply taking only 3 cycles.

There is a 32 Kilobyte (KB), on-die, Level 1 cache comprised of a 16 KB direct-mapped data cache and a 16 KB 2-way set associative instruction cache. Off-die there is a 512 KB Level 2 cache. Each Ultra has 128 MB of RAM and two 1 GB local hard drives. The I/O bus operates at 25 MHz and has a 64 Bit wide data path.

K.2 IBM SP2

The IBM SP2 is an MPP with a multistage Omega network. Each group of eight (8) nodes are connected via a switch board, called a frame, that is comprised of four 4x4 omega switches. Multiple frames are interconnected to scale the network with intermediate switching hardware.

The IBM SP2, located at the Aeronautical System Center's (ASC) Major Share Resource Center (MSRC) at Wright-Patterson AFB, has 256⁷⁹ nodes [83]. Each node is a 135 MHz RS/6000 Power2 SC (P2SC) processor with 1 GB of RAM. The P2SC is a four-issue superscalar processor that can perform two simultaneous integer and FP instructions.

Each processor has a 128 KB 4-way set associative data cache and a 32 KB instruction cache. The network interface card (NIC) on each node has a Power PC 601 processor and performs DMA only to and from the host processor. The DMA performance varies with a maximum transfer rate of 160 MB/sec on the 64 bit 20 MHz micro-channel bus. The network has a peak theoretical bandwidth of 300 MB/sec in full-duplex mode.

⁷⁸ The Myrinet is capable of either 1.28 Gbit or 2.56 Gbit full-duplex communications.

⁷⁹ Only 233 nodes are available for processing.

K.3 AFIT Heterogeneous Beowulf

The AFIT ABC Beowulf consists of one DELL 450 MHz Pentium⁸⁰ II processor, six DELL 400 MHz Pentium II processors, and four Gateway 333 MHz Pentium II processors connected via a 100 Mb/sec full duplex switched Fast Ethernet. Each processor can be booted with either Windows NT 4.0 or Linux 2.0.33. The two operating systems mounted on separate hard drives. Parallel communication is handled through MPI/PRO 1.2.3 or Patent MPI 4.0 for Windows NT and MPICH version 1.1 for Linux applications.

Three of the four Gateways have 128 Mb 15 nsec SDRAM, and each of the DELL processors has 128 MB of 10 nsec⁸¹ SDRAM. The fourth Gateway has 256 Mb 15 nsec SDRAM. The Pentium II processor Level 1 cache consists of a 4-way set associative 16 KB instruction cache and 16 KB nonblocking 2-way set associative dual ported data cache. The Level 2 cache is 512 KB nonblocking, squashing, unified 4-way set associative physically addressed L2 cache capable of handling four outstanding misses and has a twelve entry load queue. The L2 cache is clocked at half the speed of the processor.

Under the NT configuration, each Gateway processor has one 8 GB EIDE hard drive at its disposal; the DELL computers each have one 8.4 GB SCSI hard drive. When the system is Linux, each processor (Gateway or DELL) has one 5.6 GB EIDE hard drive available, except one that has an 540 MB EIDE hard drive.

Finally, the I/O bus on the Gateways operates at 66 MHz whereas the DELL's I/O bus is clocked at 100 MHz.

⁸⁰ Pentium II is a registered trademark of the Intel Corporation.

⁸¹ nsec = nanoseconds or 10^{-9} seconds

8.0 Vita

Captain Karl R. Deerman enlisted in the United States Air Force in June 1989. He attended the USAF Preparatory School. Upon graduation, he attended the USAF Academy where he earned his bachelor's degree in computer science and received his commission in June 1994. He was assigned to the Air Force Operational Test and Evaluation Center (AFOTEC), New Mexico, where he was responsible for the test and evaluation of the software for the B-2, the Unmanned Aerial Vehicle (UAV), the Joint Computer-based Acquisitions and Logistic System. Captain Deerman left the AFOTEC in 1998 to attend AFIT. He was subsequently assigned to Air Force Research Laboratory where he will apply his education to similar research projects.

9.0 References

- [1] Goldberg, David E.; Korb, Bradley; Deb, Kalyanmoy; "Messy Genetic Algorithms: Motivation, Analysis, and First Results" Complex Systems Publications, 1989.
- [2] Goldberg, David E.; Korb, Bradley; Deb, Kalyanmoy; "An Investigation of Messy Genetic Algorithms" The Clearing House for Genetic Algorithms, University of Alabama. May 1990.
- [3] Lamont, Gary, et al., "AFIT Compendium of Genetic Algorithms"
- [4] Corno, F.; Reorda, Sonza; and Squillero, G.; "A New Evolutionary Algorithm Inspired by the Selfish Gene Theory" Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy
- [5] Corno, F.; Reorda, Sonza; and Squillero, G.; "The Selfish Gene Algorithm: a new Evolutionary Optimization Strategy" Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy.
- [6] Whitley, Darrell; "A Genetic Algorithm Tutorial" Computer Science Department, Colorado State University.
- [7] Harik, Georges R.; Goldberg, David E.; "Learning Linkage" Department of General Engineering, University of Illinois at Urbana-Champaign. August 1996.
- [8] Goldberg, David E.; Harik, Georges R.; Lobo, Fernando; Deb, Kalyanmoy; and Wang, Liwei; "Compressed Introns in a Linkage Learning Genetic Algorithm" Department of General Engineering, University of Illinois at Urbana-Champaign; December, 1997.
- [9] Kargupta, Hillol; "SEARCH, Evolution, and The Gene Expression Messy Genetic Algorithm," Computational Science Methods Division, Los Alamos National Laboratory, Los Alamos, NM.
- [10] Kargupta, Hillol; Bandyopadhyay, Sanghamitra; Wang, Gang; "Revisiting the GEMGA: Scalable Evolutionary Optimization Through Linkage Learning," Department of Computer Science Washington State University at Pullman.
- [11] Kargupta, Hillol; "The Gene Expression Messy Genetic Algorithm." Published in the Proceedings of the IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996.
- [12] Harik, Georges; Lobo, Fernando; Goldberg, David; "The Compact Genetic Algorithm" Department of General Engineering, University of Illinois at Urbana-Champaign. 1997.
- [13] Beasley, David; Bull, David; Martin, Ralph; "An Overview of Genetic Algorithms: Part 1, Fundamentals" Inter-University Committee on Computing, 1993.

- [14] Merkle, Laurence, Generalization and Parallelization of Messy Genetic Algorithms and Communication in Parallel Genetic Algorithms. MS Thesis, AFIT/GCE/ENG/92-D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1992.
- [15] Gates, George Jr., Predicting Protein Structure Using Parallel Genetic Algorithms. MS Thesis, AFIT/GCS/ENG/95-D, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1994.
- [16] Back, Thomas, Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press Inc., New York, 1996.
- [17] Harik, Georges R., "Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms" Phd. Dissertation, The University of Michigan, Ann Arbor, MI, 1997.
- [18] Brinkman, Donald J., Genetic Algorithms and Their Application to the Protein Folding Problem. MS Thesis, AFIT/GCS/ENG/93-D, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1993.
- [19] Sober, Herbert A., CRC Handbook of Biochemistry. The Chemical Rubber Co. Cleveland, OH, 1968.
- [20] Schnieder, Michael, "PSC Simulation Tracks Folding Proteins", HPCwire, 28 Aug 98, URL: hpcwire@tgc.com.
- [21] "Stereochemistry of Polypeptide Chain Configurations", Journal of Molecular Biology, vol. 7, 1963, pg. 95 – 99.
- [22] Stryer, Lubert, Biochemistry: Fourth Edition. W. H. Freeman and Company, New York, NY, 1995.
- [23] "An Introduction to Magnetic Resonance" URL: [www.theochem.uni-
duisburg.de/PC/NMR/Theory/nmr_bloc.doc](http://www.theochem.uni-duisburg.de/PC/NMR/Theory/nmr_bloc.doc).
- [24] "Nuclear Magnetic Resonance Spectrometer" URL: [www.brevard.cc.fl.us/BTR-
Labs/facils/instrums/nmr.htm](http://www.brevard.cc.fl.us/BTR-Labs/facils/instrums/nmr.htm).
- [25] "Nuclear Magnetic Resonance" URL: www.mmi.org/mmi/standard/anser/nmr.html.
- [26] Picture from URL: [www.ems.uwplatt.edu/sci/chem/fac/sundin/363-
7/image/nmr~c001.gif](http://www.ems.uwplatt.edu/sci/chem/fac/sundin/363-7/image/nmr~c001.gif).
- [27] "X-ray Crystallography" URL: www.iumsc.indiana.edu/xray.htm.
- [28] "Pardue: Behind the Science, X-ray Crystallography." URL: www.pardue.edu/UNS/html4ever/9804/crystallography.html.

- [29] URL: <http://www-wilson.ucsd.edu/education/xraydiff/59.10B.jpeg>.
- [30] Computational Engines Synopsis. URL: cmm.info.nih.gov/modeling/gateway.html.
- [31] Molecular Mechanics Background. URL: cmm.info.nih.gov/modeling/guide_documents/molecular_mechanics_document.html.
- [32] Introduction to Hartree-Fock Calculations. URL: www.fys.ruu.nl/~west/Verslag/node5.html.
- [33] Quantum Chemistry. URL: cmm.info.nih.gov/modeling/guide_documents/quantum_mechanics_document.html.
- [34] Schrodinger's Equation Outline. URL: cobra.aml.arizona.edu/tutorials/schrodinger/4.html.
- [35] Schrodinger's Equation: Constantly Changing Potentials. URL: iluvatar.ncssm.edu/pages/pysics/modern/schrodinger.html.
- [36] Kumar, Vipin; Grama, Ananth; Gupta, Anshul; Karypis, George; Introduction to Parallel Computing: Design and Analysis of Algorithms. The Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.
- [37] Kaiser, Charles Jr., Refined Genetic Algorithms for Polypeptide Structure Prediction. AFIT/GCS/ENG/96D-13, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1996.
- [38] "Coupled CHARMM." URL: www.psc.edu/science/projects.html.
- [39] Drodty, Vincent; Sawyer, George; Lamont, Gary; "Introduction To Formal Parallel Design Using UNITY" AFIT Compendium, 1993.
- [40] Chandy; Misra; Parallel Program Design. Addison Welsey, 1988.
- [41] "Comparison of Force Fields," cmm.info.nih.gov/modeling/guide_documents/molecular_mechanics_document.html.
- [42] Brooks, Bernard; Bruccoleri, Robert, Olafson, Barry; States, David; Swaminathan S., Karplus, Martin. "CHARMM A Program for Macromolecular Energy, Minimization, and Dynamics Calculations" Journal of Computational Chemistry, Vol 4, No 2, John Wiley and Sons, INC. 1983, pg. 187-217.
- [43] Shirazi, Behrooz; Hurson, Ali R.; "Parallelism Management: Scheduling and Load Balancing" AFIT CSCE790 class notes Summer 1998.
- [44] Toran, Jacobo, "Chapter 9: P-Completeness".
- [45] Kargupta, Hillol; Goldberg, David; Wang, Liwei; "Extending the Class of Order-k Delineable Problems for the Gene Expression Messy Genetic Algorithm." To be

published in the proceedings of the Genetic Programming Conference, AAAI Press, Stanford, 1997.

[46] Table supplied by Capt Van Veldhuizen.

[47] Committee on Physical, Mathematical, and Engineering Sciences. "Grand Challenges 1993: High Performance Computing and Communications." Office of Science and technology Policy, 1992.

[48] Crescenzi; Goldman; Papadimitriou; Piccoloboni; Yannakakis; "On the Complexity of Protein Folding"

[50] Neumaier, Arnold; "Molecular Modeling of Proteins and Mathematical Prediction of Protein Structure" Society for Industrial and Applied Mathematics Review, volume 39, number 3, Sep 97.

[51] Kloeppel, James; "Fast Measurement Technique Reveals Early Steps in Protein Folding" News from the University of Illinois at Urbana-Champaign, Dec 96.

[52] Kaiser, Charles; Lamont, Gary; Merkle, Laurence; Gates, George; Pachter, Ruth; "Polypeptide Structure Prediction: Real-Value Versus Binary Hybrid Genetic Algorithms" ACM Symposium on Applied Computing, Feb 1997.

[54] Yao, Xin; Lui, Yong; "Fast Evolutionary Strategies" University of New South Wales, Australia.

[55] Chellapilla, Kumar; "Evolutionary Programming Tutorial" Genetic Programming Conference 1998.

[56] Koza, John; "Introductory Tutorial on Genetic Programming" International Conference on Genetic Algorithms 1995.

[60] Nielson, Gregory; Hagen, Hans; Muller, Heinrich; Scientific Visualization Overviews, Methodologies, and Techniques. IEEE Computer Society; Los Alamitos, CA. 1997.

[61] Ramachandran; Ramakrishnan, and Sasisekharan "Stereochemistry of Polypeptide Chain Configurations." Journal of Molecule Biology, Volume 7, pgs 95 – 99, 1963.

[62] Bindewald, Eckart; Hesser, Jurgen; Manner, Reinhard; "Implementing genetic Algorithms with Sterical Constraints for Protein Structure Prediction" University of Mannheim, Mannheim Germany, 1997

[63] Creighton, Thomas E.; Proteins: Structures and Molecular Properties. Edition 2, W. H. Freeman and Company, New York, NY, 1993.

[64] Brooks, Charles; Gruebele, Martin; Onuchic, Jose; Wolynes, Peter; "Chemical Physics of Protein Folding" Proceeding National Academy of Science. Vol 95, pp 11037-11038, Sep 1998.

[65] Darnell, James; The Processing of RNA" Scientific America, volume 253, pp. 90 – 97, Oct 1985.

[66] Dymek, Andrew; Examination of Hypercube Implementations of Genetic Algorithms. AFIT/GCS/ENG/92M-02, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, March 1992.

[67] Oparin, Alxeander; hkusuc.hku.hk/physics/public_html/seti/15origin.html.

[70] Page-Jones, Meilir; The Practical Guide to Structured Systems Design. Yourdon Press; Englewood Cliffs, New Jersey; 1988.

[71] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Fredrick; Lorensen, William; Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[72] Pacheco, Peter; Parallel Programming with MPI. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[73] Gaulke, Robert; The Application of Hybridized Genetic Algorithms to the Protein Folding Problem. AFIT/GCS/ENG/95D-03, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1995.

[74] Arnold, Allen; Probability, Statistics, and Queuing Theory with Computer Science Applications. 2nd Edition. Academic Press, Inc. Boston, Mass. 1990.

[75] Barr, Richard; Golden, Bruce; Kelly, James; Resende, Mauricio; Stewart, William Jr.; "Designing and Reporting on Computational Experiments with Heuristic Methods" Journal of Heuristics. Kluwer Academic Publishers, 1995.

[76] Bradley, Ralph; Kendall, David; Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., New York, NY, 1978.

[77] Fogiel, Max; REA's Problem Solvers Statistics: A complete Solution Guide to Any Textbook. Research and Education Association. New Jersey, 1998.

[78] Gindhart, David, A Comparative Analysis of Networks of Workstations and Massively Parallel Processors for Signal Processing. AFIT/GCE/ENG/97D-01, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1997.

[79] Schulze-Kremer, Steffen, BioComputing Hypertext CourseBook: Chapter 5. www.techfak.uni-bielefeld.de/bcd/Curric/ProtEn/protein.html

[80] Piccolboni, A.; Mauri, G.; "Distance Space Evolutionary Algorithms for Protein Folding Predictions" IEEE 1998.

[81] Samudrala, Ram; Moulton, John; "A Graph-theoretic Algorithm for C0omparative Modeling of Protein Structure" BIOINFORMATICS<--> STRUCTURE, 17-21 Nov 1996

[82] Gindhart, David; A Comparative Analysis of Networks of Workstations and Massively Parallel Processors for Signal Processing. AFIT/GCE/ENG/97D-01, School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB, OH, December 1997.

[83] "IBM SP User's Guide" Aeronautical Systems Center Major Share Resource Center, Wright-Patterson AFB, OH. www.asc.hpc.mil.

[84] Thakur, Rajeev; Gropp, William; Lusk, Ewing; "A Case for Using MPI's Derived Datatypes to Improve I/O Performance" Super Computing 1998.

[85] Box, George; Hunter, William; Hunter, Stuart; Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building. John Wiley and Sons, New York, NY, 1978.

[86] Tavares, Rui; Teofil, Antonio; Silva, Paulo; Rosa, Agostinho; "Infected Genes Evolutionary Algorithm" SAC 99, San Antonio, TX.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 23 March 1999		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE PROTEIN STRUCTURE PREDICTION USING PARALLEL LINKAGE INVESTIGATING GENETIC ALGORITHMS			5. FUNDING NUMBERS	
6. AUTHOR(S) Karl R. Deerman, Captain, USAF				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2950 P Street, Bldg 640 WPAFB OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/99M-03	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Ruth Pachter AFRL/MLPJ 2941 P Street, Bldg 651 Wright-Patterson AFB, OH 45433-7702 COMM: (937) 255-6671x3158 DSN: 785-6671 x3158			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Dr. Larry B. Lamont COMM: (937) 255-3636 x4718 DSN: 785-3636 x4718				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This is an engineering investigation into the effectiveness and efficiency of the Linkage Learning GA (LLGA) applied to the PSP problem. The LLGA implementations takes explicit advantage of "tight linkages" early enough in its algorithmic processing to overcome the disruptive effects of crossover. The LLGA is integrated with the previously developed and tested AFIT CHARMM energy model software. Furthermore, a parallel version, pLLGA, is developed using a data partitioning scheme to "farm out" the CHARMM evaluations. Portability across AFIT's heterogeneous ABC Beowulf system, distributed networks, and massively parallel platforms is accomplished through the use of object-oriented C++ and the Message Passing Interface (MPI). This model improves the efficiency of the LLGA algorithm. Ramachandran developed constraints are incorporated into the LLGA to exploit domain knowledge in order to improve the effectiveness of the search technique. This approach, constrained-LLGA (cLLGA), has been parallelized using the same decomposition as the pLLGA. This new implementation is called the constrained-parallel LLGA (cpLLGA). Efficiency analysis for these two implementations is discussed. Finally, the results from these experiments are compared to previous AFIT implementations. The parallel fast messy GA and the parallel real-valued GA are compared to the pLLGA and cpLLGA, respectively.				
14. SUBJECT TERMS GENETIC ALGORITHMS, PROTEIN STRUCTURE PREDICTION, PARALLEL, LINKAGE LEARNING, MPI, CHARMM, RAMACHANDRAN CONSTRAINTS, VISUALIZATION, BEOWULF			15. NUMBER OF PAGES 213	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED		18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED		19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED
				20. LIMITATION OF ABSTRACT UL